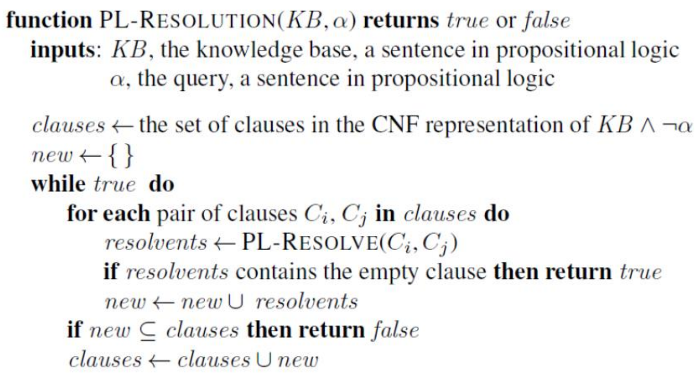
Com S 572 Fall 2020

Project B: A Theorem Prover for Propositional Logic

Summary

- Pass KB sentences as string and analyze string. Parse string from infix to postfix in order to construct expression tree.
- Infix2Postfix.java includes precedence of operator in order to use with stack.
- Then expression tree is used to compute CNF from the bottom to top (post order). The root CNF is the final Conjunctive Normal form expression
- ExressionTree.java recursively traverse through each node in the tree and keep updating node CNF value.
- Each node can be operator or operands which do different actions.
    - '~' only has right child. All other operators have both right, and left children
    - If node is operand, simply add to the node CNF clause
    - If node is operator, process children node and get this.CNF.
        - '~': apply negationFunction. The most difficult function to implement.
        - '|': apply orFunction
        - '=': applu implyFunction
        - '<': apply biImplyFunction
        - '&' apply andFunction
- Literal.java: literal is represented by char (variable value) and a boolean value that shows whether its ~ or not. it includes functions that can compare with different literals with or without literal sign.
- Clause.java is mainly a list of Literal which include add(Clause c), add(Literal l), equals(Clause c) if every literal and literal sigh is identical.
- ConjunctiveNormalForm.java is mainly a list of clause.
- During transformation process, we have to do calculation manually and then consider the pattern and think how to put result in the form of CNF(c1{l1,l2}, c2{l1,l2} …..) by using exsiting data structures. Maybe set is better than lists. I felt complicated by using list in this project.
- So far, we can express each sentence by root CNF
- Next step is applying resolution rule to see if we can get goal sentence



```
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 ← the set of clauses in the CNF representation of $KB \wedge \neg\alpha$
  new ← { }
  while true do
     for each pair of clauses $C_i, C_j$ in clauses do
        resolvents ← PL-RESOLVE($C_i, C_j$)
        if resolvents contains the empty clause then return true
        new ← new ∪ resolvents
     if new ⊆ clauses then return false
     clauses ← clauses ∪ new
```

PLResolution(KB, L) takes all CNF from KB and one specified goal into one clause list. Process this clause list to see whether newly generated resolvent by PLResolvetest is empty. If so return true representing goal can be entailed by given KB. if not empty and not null, then add to new set. if new set is subst of allClause, then return false means that cannot entail the goal literal.

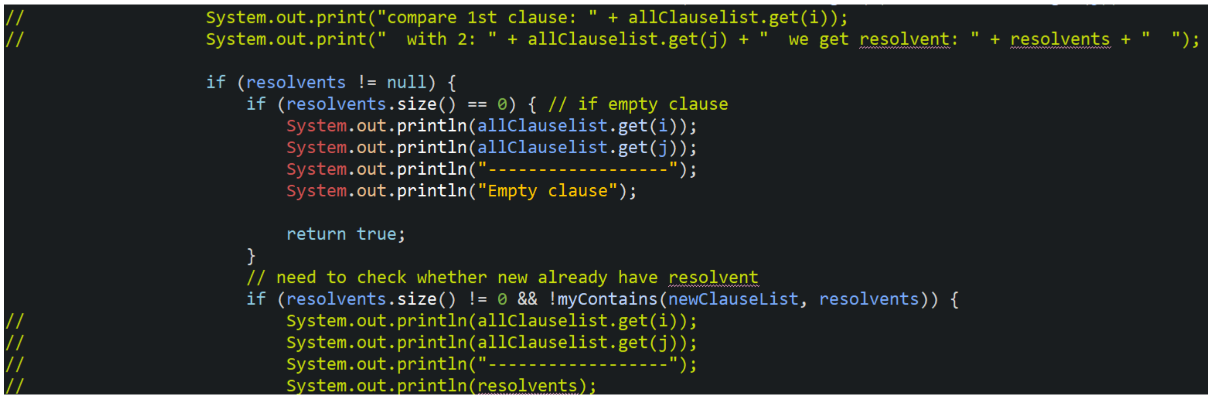
PLResolve(clause, clause) apply the resolution rule and return resolvent.

Note:

- one difficult point is to compare list of clauses with another list of clauses. Because two list can have different order of clause, and clause can contains different literals and also literals' sign. Literal is also an object and therefore have to implement its special equal() function.
- Only apply resolution rule to those clauses that have opposite literal sign. if two clauses do not have opposite literal sign, then return null, if have, but after apply rule, their size is 0, then return empty set, else, delete literal with opposite sign and "or" the rest literals and return new clause.
- Compare every pair of clause list: (x,y,z) -> (x,y), (y,z), (x,z)
- When we adding new resolvent to new set, we need to make sure not including clause that is already existed in new set.
- Same for allClauses, the new adding new set to allClause list, we have to make sure adding only the clause that allClause does not have. This is one of the time-consuming part in my program.

For simplicity and the purpose of easy to read, also due to the structure of my program, I comment out "System.out.println("------------------");" to remove all inner steps.

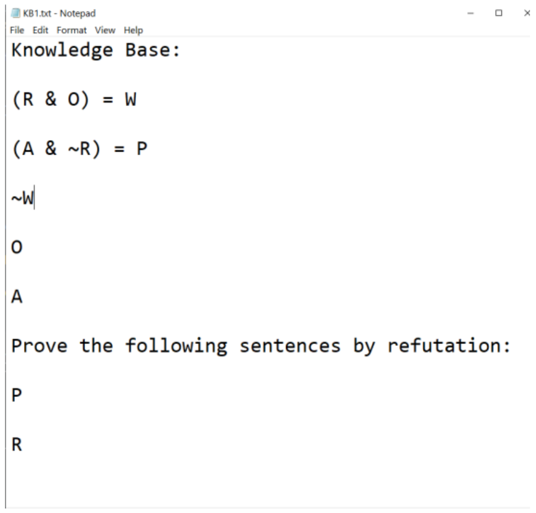
To check inner step, simply re-comment those lines in main.java program:



```
//                  System.out.print("compare 1st clause: " + allClauselist.get(i));
//                  System.out.print("  with 2: " + allClauselist.get(j) + "  we get resolvent: " + resolvents + "  ");

                    if (resolvents != null) {
                        if (resolvents.size() == 0) { // if empty clause
                            System.out.println(allClauselist.get(i));
                            System.out.println(allClauselist.get(j));
                            System.out.println("------------------");
                            System.out.println("Empty clause");

                            return true;
                        }
                        // need to check whether new already have resolvent
                        if (resolvents.size() != 0 && !myContains(newClauseList, resolvents)) {
//                          System.out.println(allClauselist.get(i));
//                          System.out.println(allClauselist.get(j));
//                          System.out.println("------------------");
//                          System.out.println(resolvents);
```

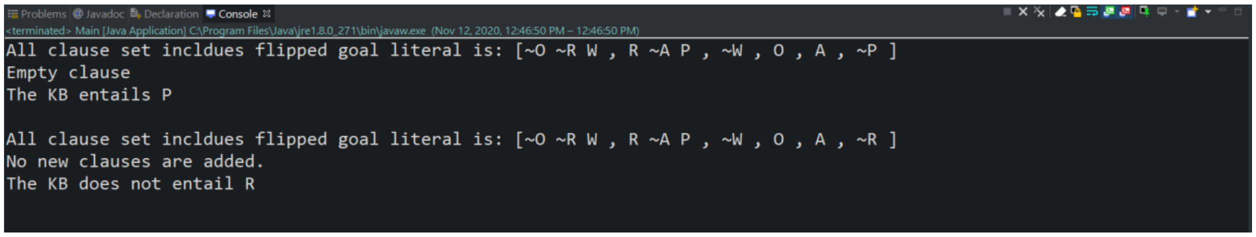
Screen shots:

KB.txt file:



```
Knowledge Base:

(R & O) = W

(A & ~R) = P

~W

O

A

Prove the following sentences by refutation:

P

R
```

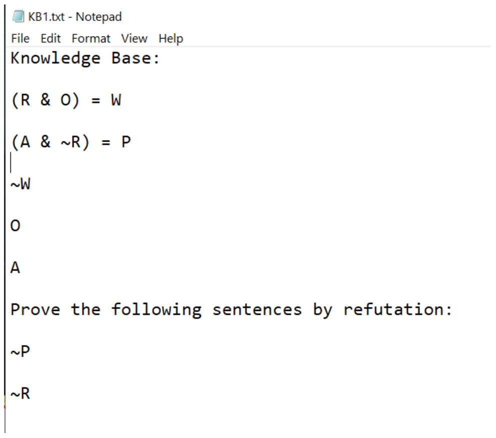
Console output:



```
<terminated> Main [Java Application] C:\Program Files\Java\jre1.8.0_271\bin\javaw.exe  (Nov 12, 2020, 12:46:50 PM – 12:46:50 PM)
All clause set incldues flipped goal literal is: [~O ~R W , R ~A P , ~W , O , A , ~P ]
Empty clause
The KB entails P

All clause set incldues flipped goal literal is: [~O ~R W , R ~A P , ~W , O , A , ~R ]
No new clauses are added.
The KB does not entail R
```

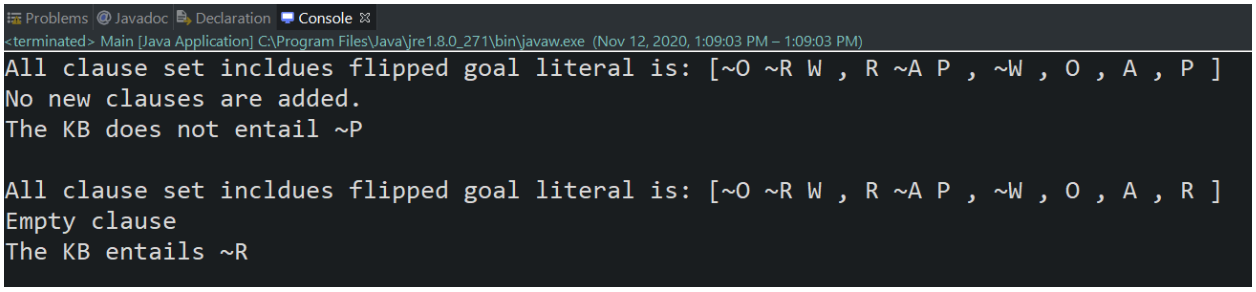
As showing below, when we negate P, and R get ~P, and ~R, program cannot entail ~P but can entail ~R. which verifies the conclusion above.

KB.txt file:



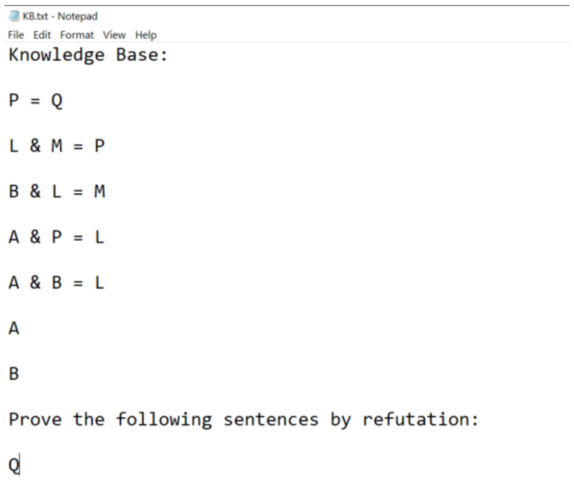
```
Knowledge Base:

(R & O) = W

(A & ~R) = P

~W

O

A

Prove the following sentences by refutation:

~P

~R
```

Console output:



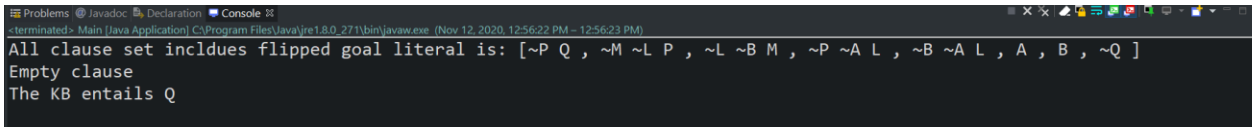
```
<terminated> Main [Java Application] C:\Program Files\Java\jre1.8.0_271\bin\javaw.exe  (Nov 12, 2020, 1:09:03 PM – 1:09:03 PM)
All clause set incldues flipped goal literal is: [~O ~R W , R ~A P , ~W , O , A , P ]
No new clauses are added.
The KB does not entail ~P

All clause set incldues flipped goal literal is: [~O ~R W , R ~A P , ~W , O , A , R ]
Empty clause
The KB entails ~R
```

KB.txt file:



```
Knowledge Base:

P = Q

L & M = P

B & L = M

A & P = L

A & B = L

A

B

Prove the following sentences by refutation:

Q
```

Console output:



```
<terminated> Main [Java Application] C:\Program Files\Java\jre1.8.0_271\bin\javaw.exe  (Nov 12, 2020, 12:56:22 PM – 12:56:23 PM)
All clause set incldues flipped goal literal is: [~P Q , ~M ~L P , ~L ~B M , ~P ~A L , ~B ~A L , A , B , ~Q ]
Empty clause
The KB entails Q
```

Reference:

infix to postfix: https://www.geeksforgeeks.org/stack-set-2-infix-to-postfix/

expression tree: https://www.geeksforgeeks.org/expression-tree/