ΟΙΚΟΝΟΜΙΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ
ΤΕΧΝΗΤΗ ΝΟΗΜΟΣΥΝΗ
2Η ΕΡΓΑΣΙΑ

ΜΕΛΗ:
• *ΣΠΥΡΙΔΟΥΛΑ ΒΛΑΤΑΚΗ*
***ΑΜ**:3150012*
• *ΚΩΣΤΑΝΤΙΝΟΣ ΜΕΣΣΑΝΑΚΗΣ*
***ΑΜ**:3150107*
• *ΓΕΩΡΓΙΑ ΧΑΤΖΗΛΥΓΕΡΟΥΔΗ*
***ΑΜ**:3150223*

ΜΕΡΟΣ ΠΡΩΤΟ :

ΕΞΑΓΩΓΗ ΣΥΜΠΕΡΑΣΜΑΤΩΝ ΣΤΗΝ ΠΡΟΤΑΣΙΑΚΗ ΛΟΓΙΚΗ ΜΕ ΧΡΗΣΗ ΤΟΥ ΚΑΝΟΝΑ ΤΗΣ ΑΝΑΛΥΣΗΣ.

Σημειώνω ότι :
Για το συγκεκριμένο μέρος χρησιμοποιήσαμε στο μεγαλύτερο μέρος τον κώδικα του εργαστηρίου που μας είχε δοθεί .

Αρχικά στο txt με όνομα pl_resolution έχουμε αποθηκεύσει την βάση γνώσης (είναι αυτή που αντιστοιχεί στο κόσμο του Wumpus από τις διαφάνειες →
(!p21 V b11)^(b11 V p12 V p21)^(!p12 V b11)^(b11))
Πέρα από τις κλάσεις του εργαστηρίου
δημιουργήσαμε και μία κλάση με όνομα Resolution στην οποία πραγματοποιούνται τα παρακάτω.

Ζητάμε από το χρήστη να πατήσει το path του txt από όπου πρέπει να διαβάσει τη βάση γνώσης .

‘Υστερα μετατρέπουμε τα δεδομένα της βάση γνώσης που διαβάζουμε στην μορφή που θέλουμε.

```
String [] subC = KBline.split("\\^");
```

Χωρίζουμε τη βάση γνώσης ,σε sub clauses όπου συναντάμε τη σύζευξη.

Κόβουμε τις παρενθέσεις και τα κενά από κάθε sub clause

```
for (int i = 0; i < subC.length; i++) {
    subC[i] = subC[i].trim();
    subC[i] = subC[i].substring(1, subC[i].length() - 1);
    subC[i] = subC[i].trim();

     subL = subC[i].split("V")
```

Αποθηκεύουμε σε μία μεταβλητή τα sub clauses χωρισμένα και με τις διαζεύξεις.

Ελέγχω αν έχει ! (δηλαδή άρνηση) μπροστά και αν ναι :

```
l.getLiterals().add(new Literal(temp.substring(1),true))
```

Περνάω true αν είναι άρνηση

Δημιουργούμε ένα Clause που περιέχει όλα τα SubClauses της βάσης γνώσης αλλά και το αρνητικό αυτού που θέλουμε να αποδείξουμε.Η υλοποίηση του κανόνας της ανάλυσης γίνεται με τον παρακάτων τρόπο.(περιεκτικά)

```
for(int i = 0; i < subclauses.size(); i++)
        {
           CNFSubClause Ci = subclauses.get(i);

           for(int j = i+1; j < subclauses.size(); j++)
           {
              CNFSubClause Cj = subclauses.get(j);

           Vector<CNFSubClause> new_subclauses_for_ci_cj = CNFSubClause.resolution(Ci,Cj);
              for(int k = 0; k < new_subclauses_for_ci_cj.size(); k++)
              {
                 CNFSubClause newsubclause = new_subclauses_for_ci_cj.get(k);}

      if(newsubclause.isEmpty())
```

Παίρνει κάθε SubClause και τα συγκρίνει με καθένα από αυτά που υπάρχουν στην βάση γνώσης .Καλεί την συνάρτηση resolution η οποία θα σου επιστρέψει τα literals και των δύο εκτός από το ζευγάρι που είναι η άρνηση του καθενός.

Επιστρέφει ένα vector με τα παραπάνω literlas.

Ελέγχει αν έχουμε φτάσει στη κενή διάζευξη

<u>ΜΕΡΟΣ ΔΕΥΤΕΡΟ</u> :

ΕΞΑΓΩΓΗ ΣΥΜΠΕΡΑΣΜΑΤΩΝ ΣΤΗΝ ΠΡΟΤΑΣΙΑΚΗ ΛΟΓΙΚΗ ΜΕ ΠΡΟΤΑΣΕΙΣ HORN

Αρχικά στο txt με όνομα fc_horn έχουμε αποθηκεύσει την βάση γνώσης (παράδειγμα από διαφάνειες)
→
P=>Q
L^M=>P
B^L=>M
A^P=>L
A^B=>L
A
B

Η κλάση που υλοποιεί τη παραπάνω μέθοδο ονομάζεται Horn.
<u>Αποθηκεύουμε</u> :
Σε μία ArrayList ***agenda*** *τα γεγονότα που έχουμε στην ΒΓ χωρίς να έχουμε διερευνήσεις τις συνεπειές τους.*
!(line=KB.get(i)).contains("=>")

Δηλαδή όλα αυτά που δεν έχουμε διερευνήσει τις συνεπειές τους.

Σε μία ArrayList τα ***horn clauses*** *δηλαδή πιο απλά όλους τους τύπους που βρίσκονται αριστερά της συνεπαγωγής.*

Σε μία ArrayList ***str_inferred*** *όλα τα γεγονότα που βρίσκονται μετά το συνεπάγεται αλλά και τα γεγονότα που έχουμε στην agenda και βρίσκεται σε αντιστοιχεία με την* ***inferred*** *η οποία είναι τύπου boolean και μας δείχνει αν έχουμε συμπεράνει το κάθε γεγονός ή όχι.*

Σε μία ArrayList ***count*** *τον* αριθμό συνθηκών του κανόνα που απομένει να ικανοποιηθούν , δηλαδή ανάλογα με τις συζεύξεις που υπάρχουν πριν το συνεπάγεται , θέλουμε κι αντίστοιχο αριθμών γεγονότων που πρέπει να έχουμε συμπεράνει για να συμπεράνουμε το γεγονός που θέλουμε.

Ο κώδικας για την υλοποίηση Horn είναι ο εξής :

```
while(!agenda.isEmpty()){
        p=agenda.remove(0);
        for(int i=0;i<str_inferred.size();i++){
           if(str_inferred.get(i)==p) pos=i;
        if(inferred.get(pos)==false){
           inferred.set(pos,true);
                        }
        for (int i=0; i<hornclauses.size(); i++){
                if (hornclauses.get(i).contains(p)){
                    count.set(i,count.get(i)-1);

                    if (count.get(i)==0){
                            inferred.set(i,true);

                      if (li_to_prove.get(i).equals(q)) {
                                    return true;
                    agenda.add(li_to_prove.get(i));
```

Εξάγω ένα στοιχείο από την agenda , και πηγαίνω να δω αν αυτό το γεγονός το έχουμε συμπεράνει ή οχι , δηλαδή αν στην arraylist inferred είναι true .

Πυροδότηση κανόνα.Ουσιαστικά βλέπω αν στο συγκεκριμένο Horn clause υπάρχει το στοιχείο που εξήγαγα ,αν ναι τότε μείωνω τον αριθμό συνθηκών δηλαδή το πόσα θέλω ακόμα να εξερευνήσω για να το συμπεράνω .Αν ο αριθμός αυτός είναι μηδέν άρα όσα έπρεπε να έχω βρεί για να τ συμπεράνω τα έχω βρει ,συνεπώς το σημειώνω ως εξερευνημένο και τσεκάρω αν αυτό που θέλαμε να αποδείξουμε είναι αυτό ,αν είναι γυρνάμε true.

<u>ΜΕΡΟΣ ΤΡΙΤΟ</u> :

ΕΞΑΓΩΓΗ ΣΥΜΠΕΡΑΣΜΑΤΩΝ ΣΤΗΝ ΚΑΤΗΓΟΡΗΜΑΤΙΚΗ ΛΟΓΙΚΗ ΜΕ ΠΡΟΤΑΣΕΙΣ HORN

Αρχικά στο txt με όνομα fol_fc έχουμε αποθηκεύσει τη βάση γνώσης (παράδειγμα από διαφάνειες)
→

American(x)^Weapon(y)^Sells(x,y,z)^Hostile(z)=>Criminal(x)
Missile(M1)
Owns(Nono,M1)
Missile(x)^Owns(Nono,x)=>Sells(West,x,Nono)
Missile(x)=>Weapon(x)
Enemy(x,America)=>Hostile(x)
American(West)
Enemy(Nono,America)

Έχουμε μία μέθοδο new_vars η οποία αλλάζει σε κάθε κανόνα τις μεταβλητές με νέες δηλαδή στο παραδειγμά μας το Missile(x) => Weapon(x) αλλάζει το x με την μεταβλητή M1, και επιστρέφει μία ArrayList με όλους τους τύπους αλλά με αλλαγμένες τις μεταβλητές δηλαδή:

Enemy(Nono,America)=>Hostile(Nono)
Missile(M1)^Owns(Nono,M1)=>Sells(West,M1,Nono)
Missile(M1)=>Weapon(M1)
American(West)^Weapon(M1)^Sells(West,M1,Nono)^Hostile(Nono)=>Criminal(West)

και μία μέθοδο getBetweenStrings την οποία καλούμε από την παραπάνω και μας επιστρέφει τη μεταβλητή που πρέπει να αλλαχθεί. Ουασιαστικά δεν έχουμε ξεχωριστή μέθοδο για την ενοποίηση, γίνεται μέσα σε αυτές τις δύο.

```
for ( int p =0; p<dif_feat.size(); p++){
  for(int l=0;l<old_clause.size();l++){
        if ( old_clause.get(l).contains(dif_feat.get(p))){
              s=dif_feat.get(p)+"(";
              str=getBetweenStrings(old_clause.get(l),s,")");
               String st1 =getBetweenStrings(facts.get(p), "(", ")");
               if ( str.contains(",")){
                     String[] sp = str.split(",");
                     String[] sp1 = st1.split(",");

              for ( int k=0; k<sp.length; k++){
                 old_clause.set(l, old_clause.get(l).replaceAll(sp[k], sp1[k]));
        }else{
               old_clause.set(l, old_clause.get(l).replaceAll(str, st1));}
               if(!res.contains(old_clause.get(l))){
                     res.add(old_clause.get(l));
```

Όταν έχει παραπάνω από δύο ορίσματα πχ Enemy(x,America)

Πχ αντικαθιστά στο παλιό Enemy(**x**,America) με Enemy (**Nono**,America)

Το προσθέτει στην ArrayList που θα επιστρέψει ,αν δν το περιέχει ήδη

Τέλος ,έχουμε την μέθοδο fol_fc_ask η οποία επιστρέφει αν βρέθηκε τελικά αυτό που ο χρήστης ζήτησε.

```
ArrayList<String> unified = new_vars(def_cl,facts);
for(int i=0;i<unified.size();i++){
      if ( !facts.contains(p = unified.get(i).split("=>")[1])){
            facts.add(p); }

            facts.removeAll(Arrays.asList(null,""));
            unified = new_vars(def_cl , facts);
            for ( int i =0; i<facts.size(); i++){
                  if ( facts.get(i).equals(a)){
                        flag = true;
      return flag;
```

Καλώ την new_vars που μου επιστρέφει τους τύπους με αλλαγμένες τις μεταβλητές

Περνάω στα γεγονότα τα καινούργια συμπεράσματα τα οποία δεν υπάρχουν

Ξανακαλώ την new_vars ν κάνει το ίδιο αλλά τώρα έχοντας και τα καινούργια συμπεράσματα

Αν αυτό που ζητάει περιέχεται τότε γυρνάω true

ΓΕΝΙΚΑ :

Στην κλάση Main έχουμε κάνει ένα μενού που ζητάει ο χρήστης ποια από τις τρεις μεθόδους θέλει να χρησιμοποιήσει και αναλόγως την απάντηση του καλούμε την συγκεκριμένη μέθοδο.

ΤΕΛΟΣ