Part B

Επιλέχθηκαν οι παρακάτω μέθοδοι εξαγωγής συμπερασμάτων:

1. Εξαγωγή συμπερασμάτων προς τα εμπρός (forward chaining) για προτάσεις Horn (ακριβέστερα, οριστικές προτάσεις) προτασιακής λογικής.
2. Εξαγωγή συμπερασμάτων προς τα εμπρός για προτάσεις Horn (ακριβέστερα, οριστικές προτάσεις) πρωτοβάθμιας κατηγορηματικής λογικής.

Προτασιακή λογική forward chaining.

Βάση προγράμματος

Υλοποιήθηκε το πρόγραμμα βασισμένο στον αλγόριθμο PL_FC_Entails της διάλεξης 9.

```
function PL-FC-ENTAILS?(KB, q) returns true or false
  local variables: count, a table, indexed by clause, initially the number of premises
                   inferred, a table, indexed by symbol, each entry initially false
                   agenda, a list of symbols, initially the symbols known to be true
  while agenda is not empty do
      p ← POP(agenda)
      unless inferred[p] do
          inferred[p] ← true
          for each Horn clause c in whose premise p appears do
              decrement count[c]
              if count[c] = 0 then do
                  if HEAD[c] = q then return true
                  PUSH(HEAD[c], agenda)
  return false
```

Γεγονότα που έχουμε στη ΒΓ χωρίς να έχουμε διερευνήσει τις συνέπειές τους (μέτωπο).

inferred[p]: Δείχνει αν έχουμε συμπεράνει το γεγονός p ή όχι.

Πυροδότηση κανόνα.

Αρχείο υλοποίησης

Ο αλγόριθμος υλοποιείται στο αρχείο PL_FC_Entails.java

<u>Βοηθητικές κλάσεις</u>

ImplicationForm.java

ReadHornFromTxt.java

Ανάλυση υλοποίησης

Στο παρακάτω τμήμα δημιουργούνται οι συλλογές και αρχικοποιούνται με τις σωστές τιμές.

Μετά την ανάγνωση από το αρχείο, η KB έχει μετατραπεί σε μορφή Implication και λόγω των οριστικών προτάσεων και της απλότητας του συγκεκριμένου προβλήματος χρησιμοποιήθηκαν μόνο String types για premises και conclusions του Implication.

Όσα έχουν currentPremise size >0 (γραμμη 25) θεωρούνται κανόνες αλλιώς γεγονότα (γραμμή 34).

```
13      public static boolean doesPL_FC_Entails(Set<ImplicationForm> KB, String q)
14      {
15          Map<ImplicationForm, Integer> count = new HashMap<>();
16          Map<String, Boolean> inferred = new HashMap<>();
17          Queue<String> agenda = new LinkedList<>();
18
19          // Initializate Collections
20          for (ImplicationForm implication : KB)
21          {
22              List<String> currentPremise = implication.getPremise();
23              String conclusion = implication.getConclusion();
24
25              if (currentPremise.size() > 0)
26              {
27                  count.put(implication, currentPremise.size());
28
29                  for (String symbol : currentPremise)
30                  {
31                      inferred.put(symbol, value:false);
32                  }
33              }
34              else // symbols known to be true because of zero premise symbols
35              {
36                  agenda.add(conclusion);
37              }
38
39              inferred.put(conclusion, value:false);
40
41          }
```

Στο κύριο μέρος εξάγουμε από την συλλογή agenda ένα στοιχείο, το συγκρίνουμε με το δοσμένο στοιχείο και επιστρέφουμε την τιμή true αν είναι ίδιο (γραμμές 46-48).

Αλλιώς συνεχίζει ο κώδικας και ελέγχει την boolean τιμή του συγκεκριμένου στοιχείου (γραμμή 52).

Αν είναι στοιχείο που δεν έχει συμπερανθεί τότε

1. Εισάγει το στοιχείο στη ανάλογη συλλογή (γραμμή 54)
2. Μειώνει το count σε όσους κανόνες περιέχουν το συγκεκριμένο στοιχείο ως premise (γραμμές 61-64)
3. Αν μετά τη μείωση κάποιος κανόνας μηδενίσει τότε το Conclusion του, επειδή είναι νέο γεγονός συγκρίνεται με το δοσμένο στοιχείο q και αν δεν είναι ίδιο, εισάγεται στη συλλογή της ατζέντας .

Ο κώδικας τερματίζει αν αδειάσει η ατζέντα και δεν έχει επιστραφεί true σε κάποια σύγκριση των γραμμών 47-48, 69-72.

```
42          // main part
43          while (!agenda.isEmpty())
44          {
45
46              String p = agenda.remove();
47              if (p.equals(q))
48                  return true;
49
50              Boolean isPInferred = inferred.get(p);
51
52              if (!isPInferred)
53              {
54                  inferred.put(p, value:true);
55
56                  for (Map.Entry<ImplicationForm, Integer> countElement : count.entrySet())
57                  {
58                      int numberOfPremises = countElement.getValue();
59
60                      // if the symbol is in the premise of current implication
61                      if (countElement.getKey().getPremise().contains(p))
62                      {
63                          count.put(countElement.getKey(), numberOfPremises - 1);
64                      }
65
66                      if (countElement.getValue().equals(0))
67                      {
68                          String conclusion = countElement.getKey().getConclusion();
69                          if (conclusion.equals(q))
70                          {
71                              return true;
72                          }
73                          agenda.add(conclusion);
74                      }
75                  }
76              }
77          }
78          return false;
```

Αποτελέσματα με δοσμένη ΒΓ διαφανειών

ΒΓ

(¬PVQ)

(¬LV¬MVP)

(¬BV¬LVM)

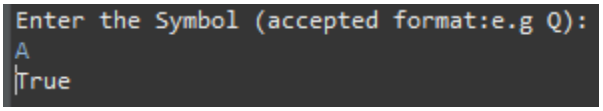
(¬AV¬PVL)

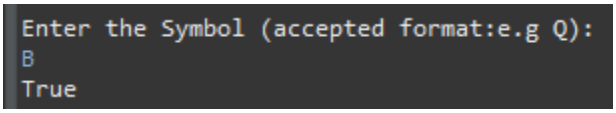
(¬AV¬BVL)

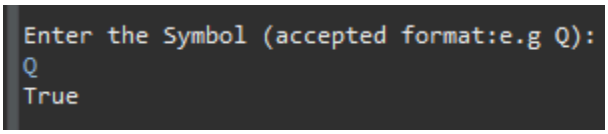
(A)

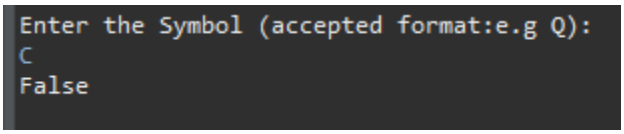
(B)

Δοσμένες ερωτήσεις και αποτελέσματα





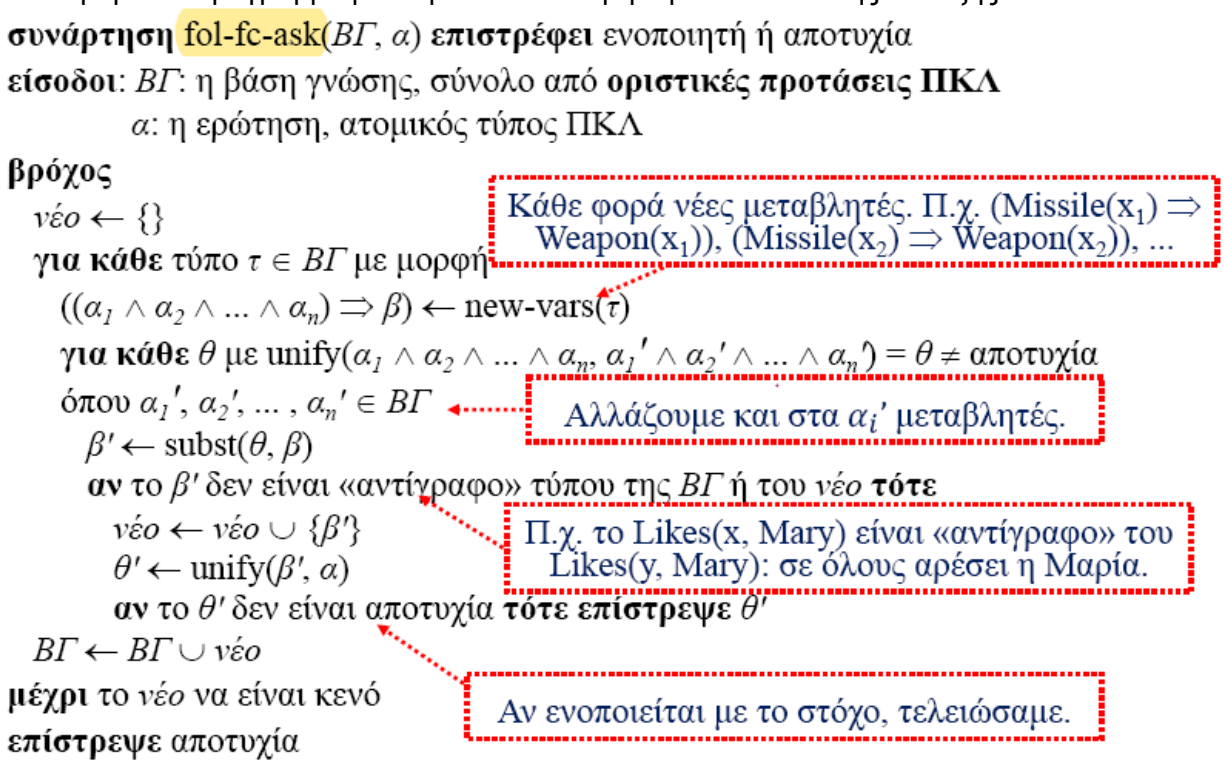




ForwardChaining_FirstOrder

Βάση προγράμματος

Υλοποιήθηκε το πρόγραμμα βασισμένο στον αλγόριθμο fol-fc-ask της διάλεξης 12.



Αρχεία υλοποίησης

Ο αλγόριθμος υλοποιείται στο αρχείο FOL.java

<u>Βοηθητικές κλάσεις:</u>

FOLLoader.java :

Implication.java

Predicate.java

Standarize.java

Term.java

Unifier.java

Ανάλυση υλοποίησης

Σε κάθε While loop το counter που αλλάζει τα ονόματα από x σε x1, x2 κλπ. μηδενίζει ώστε να μην έχουμε μεγάλους αριθμούς (γραμμή 17).

Σημείωση: τα facts είναι implications με μέγεθος λίστας `premisePredicates.`

Γραμμές 23-47: για κάθε κανόνα της KB βρίσκει μέσω της βοηθητικής μεθόδου ***`findImplicationMatch`*** αντικαταστάσεις (substitutions) για όλα τα premises συγκρίνοντάς τα με τα facts της KB. Αν υπάρχει αντικατάσταση για όλο το premise, τότε νέο implication ως fact.

```
12      public Map<String, String> fc_Ask(Set<Implication> KB, Implication a)
14          while (true)
15          {
16              // resetCounter
17              standarize.resetCounter();
18
19              Set<Implication> newSentences = new HashSet<>();
20
21              for (Implication currentImplication : KB)
22              {
23                  standarize.newVars(currentImplication);
24
25                  Map<String, String> substitutions = findImplicationMatch(currentImplication, KB);
26
27                  // there is an available substitution for current implication
28                  if (substitutions.size() > 0)
29                  {
30                      Predicate newFact = new Predicate(currentImplication.getConclusionPredicate());
31
32                      // replace variables in conclusion
33                      List<Term> conclusionTerms = newFact.getTerms();
34                      for (Map.Entry<String, String> entry : substitutions.entrySet())
35                      {
36                          for (Term term : conclusionTerms)
37                          {
38
39                              if (term.name.equals(entry.getKey()))
40                              {
41                                  term.isVariable = false;
42                                  term.name = entry.getValue();
43                              }
44                          }
45                      }
46
47                      Implication newSentence = new Implication(newFact);
```

Στη συνέχεια ελέγχεται μέσω της μεθόδου ***findPredicateMatch*** αν το νέο fact δεν μπορεί να ενοποιηθεί ούτε με κάποιο υπάρχον στοιχείο στη KB, ούτε στο `newSentences` .

Στην περίπτωση αυτή, προστίθεται στη συλλογή `newSentences` και ελέγχεται αν μπορεί να ενοποιηθεί με την πρόταση που έχει εισάγει ο χρήστης, επιστρέφοντας την αντικατασταση των μεταβλητών στην περίπτωση που ενοποιείται.

Ο κώδικας τερματίζει όταν έχει ελεγχθεί όλη η KB και δεν έχει δημιουργηθεί καμία νέα πρόταση.

```
48                  if (!findPredicateMatch(newFact, KB) && !findPredicateMatch(newFact, newSentences))
49                  {
50                      newSentences.add(newSentence);
51
52                      // if currentConclusion matches input from user
53                      if (newFact.getName().equals(a.getConclusionPredicate().getName()))
54                      {
55
56                          // check if their terms can be unified
57                          List<Term> inputTerms = a.getConclusionPredicate().getTerms();
58
59                          Unifier unifier = new Unifier();
60                          boolean canUnify = true;
61
62                          for (int i = 0; i < conclusionTerms.size(); i++)
63                          {
64                              Term currentTerm = conclusionTerms.get(i);
65                              Term otherTerm = inputTerms.get(i);
66
67                              if (!unifier.unify(currentTerm, otherTerm))
68                              {
69                                  canUnify = false;
70                                  System.out.println(x:"cant unifiy");
71                                  break;
72                              }
73                          }
74
75                          if (canUnify)
76                          {
77                              return unifier.substitutions;
78                          }
79                      }
80                  }
81              }
82          }
83
84          if (newSentences.isEmpty())
85          {
86              return null;
87          }
88
89          KB.addAll(newSentences);
```

Αποτελέσματα με δοσμένη ΒΓ διαφανειών

ΒΓ

NOTAmerican(x) OR NOTWeapon(y) OR NOTSells(x, y, z) OR NOTHostile(z) OR Criminal(x)

NOTMissile(x) OR NOTOwns(Nono, x) OR Sells(West, x, Nono)

NOTMissile(x) OR Weapon(x)

NOTEnemy(x, America) OR Hostile(x)

American(West)

Enemy(Nono, America)

Missile(M1)

Owns(Nono, M1)

Δοσμένες ερωτήσεις και αποτελέσματα

```
Enter the sentence(accepted format:e.g Criminal(x) ) :
Criminal(West)
True answer: {}
```

```
Enter the sentence(accepted format:e.g Criminal(x) ) :
Criminal(x)
True answer: {x=West}
```

```
Enter the sentence(accepted format:e.g Criminal(x) ) :
Sells(West, x, Nono)
True answer: {x=M1}
```

```
Enter the sentence(accepted format:e.g Criminal(x) ) :
Weapon(M1)
True answer: {}
```

```
Enter the sentence(accepted format:e.g Criminal(x) ) :
Hostile(Nono)
True answer: {}
```

```
Enter the sentence(accepted format:e.g Criminal(x) ) :
Criminal(John)
cant unifiy
False
```