ΑΝΑΦΟΡΑ (Β' ΜΕΡΟΣ)

Βανταράκη Αικατερίνη // 3210020

Γεωργακά Ευαγγελία // 3170023

Χασιώτη Ευαγγελία // 3120203

Forward-Chaining για προτάσεις Horn ΠΛ

Πρόγραμμα pl-fc-Entails:

Το πρόγραμμα για να τρέξει παίρνει ως argument ένα txt file.

Αν το file δεν υπάρχει βγάζει μήνυμα "No file detected."
Αν το file δεν έχει την ενδεδειγμένη σύνταξη βγάζει μήνυμα "The file does not have the correct syntax".

Ενδεδειγμένη σύνταξη:

```
# Facts
A
B

# Horn's rules
P -> Q
L & M -> P
B & L -> M
A & P -> L
A & B -> L
```

Γραμμή που περιέχει τον χαρακτήρα "#" θεωρείται σχόλιο.
Γραμμή που δεν περιέχει χαρακτήρες "#", "&" και "->" είναι facts.
Οι υποθέσεις συνδέονται με το σύμβολο "&"
Οι υποθέσεις συνδέονται με το συμπέρασμα με το σύμβολο "->"
Στον φάκελο "txtFiles" υπάρχουν αρχεία με τα οποία έγινε το debugging του προγράμματος.

****

ΠΕΡΙΕΧΟΜΕΝΑ

****

1. Η KnowledgeBaseReader διαβάζει το αρχείο που εισάγεται στο πρόγραμμα και εισάγει τα στοιχεία σε λίστες facts και rules.
   a. Μέθοδος public static void readKnowledgeBase(String filePath):
      Για κάθε γραμμή του αρχείου, αν ξεκινάει με «#» ή είναι κενή, αγνοεί τη γραμμή θεωρώντας τη ως σχόλιο. Αν η γραμμή περιέχει τους χαρακτήρες «->» την βάζει στη λίστα rules, αλλιώς τη βάζει στη λίστα facts.

2. Η KnowledgeBase δημιουργεί τη βάση γνώσης από τις λίστες facts και rules της KnowledgeBaseReader και εισάγει τα στοιχεία τους σε ένα set(fact) και ένα map(rules) μέσω των μεθόδων fillInRules και fillInFacts.
   a. Το map rules έχει ως κλειδί ένα conclusion και ως value μία λίστα με τα premises.
   b. Η μέθοδος "fillInRules" παίρνει τα rules από την ομώνυμη λίστα rules της KnowledgeBaseReader και κάνει split το string στον χαρακτήρα «->». Από το πρώτο συνθετικό παίρνει τα premises κάνοντας split στον χαρακτήρα «&» και τελικά θέτει ως κλειδί το «conclusion» και ως value μία λίστα με τα premises.
3. Η PLFCEntails περιέχει τη μέθοδο που υλοποιεί τον αντίστοιχο αλγόριθμο. Η λειτουργία της είναι η ακόλουθη:

```
// Load rules and facts
    kb.fillInRules(); // παίρνει τα rules από τη βάση γνώσης
    kb.fillInFacts(); // παίρνει τα facts από τη βάση γνώσης
    Map<String, List<String>> rules = kb.getRules(); // βάζει τα rules    σε ένα map
    Set<String> facts = kb.getFacts(); // βάζει τα facts σε set


 // Initialization of auxiliary structures
    Map<String, Integer> count = new HashMap<>(); // αρχικοποίηση map που έχει ως κλειδί
ένα conclusion και κρατάει ως value πόσα literals χρειάζονται για να πυροδοτηθεί το
conclusion.
    Map<String, Boolean> inferred = new HashMap<>(); // αρχικοποίηση map που έχει ως
κλειδί ένα conclusion και ως value ένα Boolean που αν είναι true σημαίνει ότι το συμπέρασμα
έχει εξαχθεί.

     Queue<String> agenda = new LinkedList<>(facts); // αρχικοποίηση του μετώπου
αναζήτησης με τα facts


// Data preparation – διατρέχουμε το map με τα rules.
    Για κάθε rule  σε αυτό το map παίρνουμε το κλειδί(conclusion)

Και τη λίστα με τα premises

Μετράμε πόσα literals χρειάζονται για να   πυροδοτηθεί το συμπέρασμα και τα αποθηκεύουμε
στο map count.
Κάθε συμπέρασμα αρχικά στην inferred είναι false. Επίσης

όλες οι υποθέσεις που ακόμα δεν έχουν εξαχθεί τίθενται σε false.


 // Edit agenda
Όσο το μέτωπο δεν είναι άδειο,
βγάζουμε το πρώτο στοιχείο από το μέτωπο
Εάν το σύμβολο είναι το ίδιο με το ερώτημα που έχουμε θέσει επιστρέφουμε true
Εάν από το μέτωπο έχει εξαχθεί ένα συμπέρασμα τότε αυτό τίθεται ως true
και μειώνουμε το count για το πόσες υποθέσεις χρειάζονται για να εξαχθεί το συμπέρασμα.

Εάν το count γίνει μηδέν πυροδοτείται το συμπέρασμα
```

4. Η Main παίρνει ως input argument ένα αρχείο txt και εκτελεί τον αλγόριθμο.

Παράδειγμα εκτέλεσης:

```
Give a query: Q
--------------------Initial Data----------------------------------------------
Rules: {P=[L, M], Q=[P], L=[A, B], M=[B, L]}
Agenda: [A, B]
count: {P=2, Q=1, L=2, M=2}
inferred: {P=false, Q=false, A=false, B=false, L=false, M=false}
--------------------Tree Data-------------------------------------------------
Pop up symbol: A
inferred: {P=false, Q=false, A=true, B=false, L=false, M=false}
count: {P=2, Q=1, L=1, M=2}
Agenda: [B]
--------------------Tree Data-------------------------------------------------
Pop up symbol: B
inferred: {P=false, Q=false, A=true, B=true, L=false, M=false}
Fire Up: L
count: {P=2, Q=1, L=0, M=1}
Agenda: [L]
--------------------Tree Data-------------------------------------------------
Pop up symbol: L
inferred: {P=false, Q=false, A=true, B=true, L=true, M=false}
Fire Up: M
count: {P=1, Q=1, L=0, M=0}
Agenda: [M]
--------------------Tree Data-------------------------------------------------
Pop up symbol: M
inferred: {P=false, Q=false, A=true, B=true, L=true, M=true}
Fire Up: P
count: {P=0, Q=1, L=0, M=0}
Agenda: [P]
--------------------Tree Data-------------------------------------------------
Pop up symbol: P
inferred: {P=true, Q=false, A=true, B=true, L=true, M=true}
Fire Up: Q
count: {P=0, Q=0, L=0, M=0}
Agenda: [Q]
The query 'Q' is successfully concluded
```

Backward-Chaining για προτάσεις Horn ΠΚΛ

****

ΠΕΡΙΕΧΟΜΕΝΑ

Συμβάσεις - ΒΓ :Βάση Γνώσης , ΠΚΛ : Πρωτοβάθμια Κατηγορηματική Λογική

****

Οι τύποι της ΠΚΛ που χρησιμοποιούνται είναι μόνο οριστικές προτάσεις . Είτε θα είναι ατομικές είτε συνεπαγωγές των οποίων το δεξί μέρος είναι ένα μοναδικό λεκτικό και το αριστερό μέρος είναι μία σύζευξη θετικών λεκτικών. Οι καθολικοί ποσοδείκτες εννοούνται και δεν επιτρέπονται οι υπαρξιακοί. Επομένως στη ΒΓ έχουμε:

- Facts: είναι της μορφής Predicate(x, Name) δηλαδή true -> Predicate(x,Name)
  Όπου x μεταβλητή και Name σταθερά.
- Rules: Predicate1 && Predicate && … && Predicate n -> ConclusionPredicate

Μορφή Βάσης Γνώσης:

```
American(West)
Enemy(Nono,America)
Owns(Nono,M1)
Missile(M1)


American(x) && Weapon(y) && Sells(x,y,z) && Hostile(z)->Criminal(x)
Missile(x) && Owns(Nono,x)->Sells(West,x,Nono)
Missile(x)->Weapon(x)
Enemy(x,American)->Hostile(x)
```

Κάθε γραμμή της ΒΓ έχει είτε ένα fact είτε ένα rule.

Facts: ένα μοναδικό κατηγόρημα που έχει μία λίστα από όρους.

Rule: Σύξευξη κατηγορημάτων ακολουθούμενη από το σύμβολο “->” και δεξιά του συμβόλου ένα μοναδικό κατηγόρημα.

Σύμβολο “&&”: δηλώνει σύζευξη
Σύμβολο “ ->”: δηλώνει συνεπαγωγή

Κατηγορήματα: έχουν τιμή true/false

Όροι: είναι είτε μεταβλητές είτε σταθερές.

1. Αρχείο KnowledgeBase.java

Μέλη κλάσης:

```
Map<Predicate, List<Predicate>> rules = new HashMap<>();
Set<Predicate> facts = new HashSet<>();
```

```
public void readKnowledgeBase(String path)
```

Από το δοσμένο αρχείο διαβάζει τα δεδομένα γραμμή προς γραμμή.

- ✓ Αν η γραμμή είναι κενή, αγνοείται.
- ✓ Αν η γραμμή **δεν περιέχει** τον χαρακτήρα "->", σημαίνει ότι είναι **γεγονός (fact)** και η μέθοδος **readPredicate(line)** καλείται για να διαβάσει το γεγονός και να το προσθέσει στα facts.
- ✓ Αν η γραμμή **περιέχει** τον χαρακτήρα "->", τότε πρόκειται για **κανόνα (rule)**

  Ο κανόνας χωρίζεται σε δύο μέρη με τη μέθοδο **split("->")**:

→ Το rhs μετατρέπεται σε **Predicate** και προστίθεται ως **συμπέρασμα του κανόνα**.

→ Το lhs διασπάται σε **Predicate**, που διαχωρίζονται με &&, και στη συνέχεια προστίθεται στη λίστα των **προϋποθέσεων του κανόνα**.

Το **συμπέρασμα** του κανόνα αποθηκεύεται στο map με τα rules ως κλειδί, και οι **προϋποθέσεις** του κανόνα αποθηκεύονται ως τιμή του αντίστοιχου κλειδιού.

2. Backward-Chaining Algorithm

```
Map<String, String> folBcAsk(KnowledgeBase kb, List<Predicate> goals, Map<String, String> bindingList,
Unifier unifier, Set<Predicate> visitedGoals)
```

Παίρνει ως παραμέτρους την ΒΓ, μια λίστα από στόχους που αρχικά είναι το query, το σύνολο ενοποιητών και τους στόχους που έχουμε ήδη επισκεφθεί.

Αν η λίστα από στόχους είναι κενή τότε επιστρέφεται ο τρέχων ενοποιητής. Αλλιώς παίρνουμε τον επόμενο στόχο και ελέγχουμε αν τον έχουμε ήδη επεξεργαστεί ώστε να αποφύγουμε ατέρμονη αναδρομή.

Αρχικά διατρέχουμε την λίστα με τα facts. Αν ο στόχος μπορεί να ενοποιηθεί με κάποιο fact τότε η απόδειξη έχει ολοκληρωθεί για τον συγκεκριμένο στόχο. Στη συνέχεια ελέγχουμε αναδρομικά και τους εναπομείναντες στόχους.

Εάν ο στόχος δεν βρεθεί στα facts τότε διατρέχουμε όλα τα rules μέχρι να βρούμε ένα conclusion που μπορεί να ενοποιηθεί με τον στόχο. Στην περίπτωση που βρεθεί, τότε κάθε κατηγόρημα στην λίστα με τις υποθέσεις πρέπει και αυτό να αποδειχθεί (αναδρομικά). Αποδεικνύεται εφόσον βρεθεί λίστα αντικαταστάσεων, οι οποίες εξάγονται από την βάση γνώσης. Σε αυτό το σημείο καλούμε την

```
processPremises(premises, newGoals, bindingList);
```

Η οποία κανονικοποιεί τις μεταβλητές σε όλα τα κατηγορήματα ενός κανόνα (δίνει ένα νέο μοναδικό όνομα στις μεταβλητές που είναι ίδιες) και στη συνέχεια εφαρμόζει τις αντικαταστάσεις που έχουν γίνει μέχρι εκείνο το σημείο της απόδειξης. Μετά από αυτές τις αλλαγές τα κατηγορήματα προστίθενται και αυτά στη λίστα στόχων.

Μετά το τέλος κάθε αναδρομικής κλήσης, αφαιρούνται από τους στόχους τα κατηγορήματα που αφορούσαν αυτήν την κλήση.

Σημείωση : Επειδή η εξαγωγή συμπερασμάτων στην ΠΚΛ είναι ημι-αποκρίσιμο πρόβλημα , δεν χρησιμοποιούμε σύμβολα συναρτήσεων (π.χ. FatherOf(x)) , γιατί μπορεί να μην τερματίζει πάντα ο αλγόριθμος.

3. Unifier

```
Map<String, String> substitutions;
```

Η **ενοποίηση** (Unification) εξ'ορισμού είναι μια συνάρτηση που παίρνει ως είσοδο δύο τύπους και επιστρέφει ένα σύνολο αντικαταστάσεων που αν το εφαρμόσουμε και στους δύο αυτοί ταυτίζονται. Αν τουλάχιστον ένας από τους όρους είναι μεταβλητός, πραγματοποιείται αντικατάσταση. Αν οι όροι είναι σταθερές τιμές, ελέγχει αν είναι ίσοι.

```
boolean unify(Predicate p1, Predicate p2)
```

Δύο κατηγορήματα για να ενοποιηθούν πρέπει να έχουν το ίδιο όνομα και το ίδιο πλήθος από όρους. Εφόσον αυτή η συνθήκη ικανοποιείται πρέπει κάθε όρος τους κατά αντιστοιχία να μπορεί και αυτός να ενοποιηθεί.

```
boolean unify(Term t1, Term t2)
```

Αν και οι δύο όροι t1 και t2 **δεν είναι μεταβλητές (δηλαδή είναι σταθερές), τότε απλώς συγκρίνουμε τα ονόματά τους για να δούμε αν είναι ίσα.**
Επιστρέφει **true** αν τα ονόματα είναι ίδια (δηλαδή αν οι δύο σταθερές είναι ταυτόσημες).
Αν κάποιος απ΄ο τους 2 όρους είναι μεταβλητή τότε ελέγχουμε αν αυτή μπορεί να αντικατασταθεί από τον άλλον όρο.

```
boolean substitute(String variable, String value)
```

Αρχικά ελέγχουμε αν υπάρχει ήδη κάποια αντικατάσταση για αυτήν την μεταβλητή. Αλλιώς μπορεί να υπάρχει κάποια τιμή η οποία είναι και αυτ΄ή μεταβλητή και για αυτήν υπάρχει ήδη αντικατάσταση. Αν δεν ισχύει καμία από αυτ΄΄ές τις περιπτώσεις τότε πρόκειται για μια νέα αντικατάσταση και την προσθέτουμε στο map με τις αντικαταστάσεις.
Στη συνέχεια, ανανεώνουμε το map για ήδη υπάρχουσες μεταβλητές που δεν έχουν εφαρμοσθεί ακόμη οι αντικαταστάσεις τους.

```
void substitution(String variable, String value)
```

Για παράδειγμα, αν έχουμε τον όρο Weapon(y) και τον όρο Weapon(M1),τότε {y/M1}, δημιουργώντας τη νέα πρόταση Weapon(M1).
Δηλαδή : unify(Weapon(y), Weapon(M1)) = {y/M1}

Σημείωση: Θεωρούμε ότι οι τύποι δεν έχουν κοινές μεταβλητές (standardization)

5. Παραδείγματα Εκτέλεσης

```
Give a query: Weapon(M1)
Current Goal: Predicate{name='Weapon', terms=[M1]}
Unified with Rule: Predicate{name='Weapon', terms=[x]}
Current Goal: Predicate{name='Missile', terms=[x_f68]}
Matched with Fact: Predicate{name='Missile', terms=[M1]}
YES
Substitution the proof did to answer query:
{x_f68=M1}
```

```
Give a query: American(West)
Current Goal: Predicate{name='American', terms=[West]}
Matched with Fact: Predicate{name='American', terms=[West]}
YES
Substitution the proof did to answer query:
{}
```

```
Give a query: Missile(Nono)
Current Goal: Predicate{name='Missile', terms=[Nono]}
Failed to prove: Predicate{name='Missile', terms=[Nono]}
NO
```

Ο **Backward Chaining** είναι **πλήρης** για ΒΓ που ακολουθούν *database semantics*.

- *Closed-world assumption:* οι ατομικές προτάσεις που δεν είναι γνωστό ότι είναι αληθείς είναι στην πραγματικότητα ψευδείς.
- *Domain closure:* κάθε μοντέλο δεν περιέχει περισσότερα στοιχεία πεδίου από αυτά που κατονομάζονται από τα σύμβολα σταθερών.