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

Αλαβάνος Νίκος p3130003

Γιοβανόπουλος Κωνσταντίνος p3190275

Νικηφοράκης Βασίλειος p3160114

Εισαγωγή

Η εργασία εστιάζει στην υλοποίηση μιας μεθόδου εξαγωγής συμπερασμάτων μέσω ανάλυσης (resolution) για την προτασιακή λογική. Η μέθοδος βασίζεται στην αναπαράσταση των προτάσεων σε Κανονική Συζευκτική Μορφή (CNF), διευκολύνοντας τη λογική επεξεργασία και την εξαγωγή συμπερασμάτων. Το πρόγραμμα επιτρέπει τη διαχείριση βάσεων γνώσης, την υποβολή ερωτημάτων και την απόδειξη αυτών.

Τρόπος Χρήσης του Προγράμματος

Η εφαρμογή λειτουργεί ακολουθώντας τα εξής βασικά στάδια:

1. **Φόρτωση Βάσης Γνώσης**
    - Οι λογικές προτάσεις Horn αποθηκεύονται σε ένα αρχείο κειμένου και στη συνέχεια εισάγονται στο πρόγραμμα.
    - Κάθε πρόταση διασπάται σε κατηγορήματα και όρους, ώστε να μπορεί να αναλυθεί και να χρησιμοποιηθεί στη διαδικασία επίλυσης.
2. **Υποβολή Ερωτημάτων**
    - Ο χρήστης έχει τη δυνατότητα να υποβάλει ερωτήματα μέσω της κονσόλας. Το ερώτημα μπορεί να αφορά κάποιο κατηγόρημα που βρίσκεται στη βάση γνώσης ή που μπορεί να συναχθεί από αυτή.
3. **Διαδικασία Επίλυσης**
    - Το πρόγραμμα χρησιμοποιεί τις υπάρχουσες προτάσεις της βάσης γνώσης και το ερώτημα του χρήστη για να εντοπίσει αν μπορεί να συναχθεί κάποιο συμπέρασμα.
    - Εάν προκύψει αντίφαση, τότε θεωρείται ότι το ερώτημα είναι αποδείξιμο.
4. **Απάντηση στον Χρήστη**

- Το πρόγραμμα ενημερώνει αν το ερώτημα μπορεί να αποδειχθεί, παρέχοντας το σχετικό συμπέρασμα ή αναφέροντας ότι δεν μπορεί να συναχθεί.

Δυνατότητες του Συστήματος

- Φόρτωση Βάσης Γνώσης: Υποστηρίζει τη φόρτωση προτάσεων σε CNF από αρχεία κειμένου.
- Υποστήριξη Ερωτημάτων: Δέχεται ερωτήματα και επιστρέφει αν αυτά μπορούν να αποδειχθούν.
- Εφαρμογή Resolution: Συνδυάζει υποπροτάσεις για την εξαγωγή νέων προτάσεων και τον εντοπισμό αντιφάσεων.
- Διαχείριση Αρνήσεων: Υποστηρίζει λογικές αρνήσεις, προσαρμόζοντας τη διαδικασία επίλυσης για την εξαγωγή σωστών συμπερασμάτων.

Αρχιτεκτονική Συστήματος

Το σύστημα είναι σχεδιασμένο με τη χρήση ανεξάρτητων κλάσεων που συνεργάζονται μεταξύ τους:

1. Literal
   Αναπαριστά ένα κατηγόρημα, π.χ., A ή ¬B. Περιλαμβάνει το όνομα και αν είναι λογική άρνηση.
2. CNFSubClause
   Εκπροσωπεί μια λογική υποπρόταση, όπως A ∨ ¬B. Παρέχει τη μέθοδο συνδυασμού με άλλες υποπροτάσεις.
3. CNFClause
   Περιέχει μια συλλογή από υποπροτάσεις, οργανωμένες σε λίστα.
4. KnowledgeBase
   Φορτώνει τη βάση γνώσης από αρχείο, αποθηκεύει τις προτάσεις και εκτελεί τη μέθοδο resolution.
5. Main
   Παρέχει τη διεπαφή χρήστη για την υποβολή ερωτημάτων και την επικοινωνία με την υπόλοιπη εφαρμογή.

Μέθοδος Resolution

Η μέθοδος resolution εφαρμόζεται ως εξής:

1. Μετατροπή σε CNF: Όλες οι προτάσεις μετατρέπονται σε CNF.
2. Συνδυασμός Υποπροτάσεων: Συνδυάζονται προτάσεις που περιέχουν αντιθέσεις (π.χ. A και ¬A), ώστε να παραχθούν νέες υποπροτάσεις.
3. Κενή Υποπρόταση: Αν παραχθεί κενή υποπρόταση, τότε υπάρχει αντίφαση, άρα το ερώτημα αποδεικνύεται.

Παραδείγματα Χρήσης

1. Φόρτωση Βάσης Γνώσης
Έστω το αρχείο testcase_2.txt περιέχει:

- A ∨ B
- ¬A
- ¬B

2. Ερώτημα
Ερώτημα B.

3. Διαδικασία Resolution

- Συνδυάζεται η πρόταση A ∨ B με την ¬A, παράγοντας τη νέα υποπρόταση B.
- Η πρόταση B συνδυάζεται με την ¬B, οδηγώντας σε κενή υποπρόταση.

4. Απάντηση
Το πρόγραμμα επιστρέφει true, καθώς το ερώτημα B αποδεικνύεται.

<u>Πειραματικά Αποτελέσματα</u>

Τα παρακάτω αποτελέσματα προέκυψαν από τη δοκιμή του συστήματος:

- Απλή βάση γνώσης:
  Βάση: A, ¬B.
  Ερώτημα: B.
  Αποτέλεσμα: false (το B δεν αποδεικνύεται).
- Αντίφαση στη βάση γνώσης:
  Βάση: A, ¬A.
  Ερώτημα: Οποιοδήποτε ερώτημα αποδεικνύεται λόγω της αντίφασης.
- Επαγωγική απόδειξη:
  Βάση: A ∨ B, ¬A.
  Ερώτημα: B.
  Αποτέλεσμα: true (το B αποδεικνύεται μέσω ανάλυσης).

<u>Συμπεράσματα</u>

Η μέθοδος resolution εφαρμόστηκε επιτυχώς για την εξαγωγή συμπερασμάτων στην προτασιακή λογική. Το σύστημα υποστηρίζει απλές και σύνθετες βάσεις γνώσης, παρέχοντας ακριβή αποτελέσματα.

<u>Εξαγωγή συμπερασμάτων προς τα εμπρός για προτάσεις Horn (ακριβέστερα, οριστικές προτάσεις) πρωτοβάθμιας κατηγορηματικής λογικής.</u>

Εισαγωγή

Η εργασία επικεντρώνεται στην υλοποίηση μιας μεθόδου εξαγωγής συμπερασμάτων μέσω της τεχνικής forward chaining για προτάσεις Horn στην πρωτοβάθμια κατηγορηματική λογική. Η μέθοδος βασίζεται στη διαδοχική εφαρμογή λογικών κανόνων "αν... τότε" πάνω σε δεδομένα της βάσης γνώσης, οδηγώντας στην εξαγωγή νέων συμπερασμάτων ή στην απόδειξη ενός ερωτήματος. Το πρόγραμμα υποστηρίζει τη διαχείριση σύνθετων βάσεων γνώσης, την υποβολή ερωτημάτων από τον χρήστη, και την αξιοποίηση της ενοποίησης για την επεξεργασία προτάσεων με σταθερές και μεταβλητές.

Αρχιτεκτονική Συστήματος

Το πρόγραμμα αποτελείται από τις παρακάτω κύριες κλάσεις:

1. **KnowledgeBaseImporter**
    - Διαχειρίζεται την εισαγωγή της βάσης γνώσης από το αρχείο κειμένου.
    - Επεξεργάζεται τις προτάσεις και τις μετατρέπει σε μορφή που μπορεί να χρησιμοποιηθεί από το πρόγραμμα.
2. **Predicate**
    - Αναπαριστά ένα κατηγόρημα, όπως "Owns(John, Dog)".
    - Περιέχει πληροφορίες όπως το όνομα, τους όρους που το συνοδεύουν, και αν είναι αρνημένο.
3. **Term**
    - Εκφράζει έναν όρο που περιλαμβάνεται σε ένα κατηγόρημα. Μπορεί να είναι είτε σταθερά (π.χ., "John") είτε μεταβλητή (π.χ., "x").
4. **Clause**
    - Αναπαριστά μια πρόταση ως συλλογή από κατηγορήματα.
5. **Unifier**
    - Υλοποιεί τη διαδικασία ενοποίησης, συνδέοντας μεταβλητές με σταθερές ή άλλες μεταβλητές, διασφαλίζοντας την ορθότητα των υποκαταστάσεων.
6. **Resolver**
    - Εφαρμόζει τη μέθοδο forward chaining, συνδυάζοντας προτάσεις για την εξαγωγή νέων συμπερασμάτων και εντοπίζοντας αντιφάσεις.
7. **Main**
    - Αποτελεί την κύρια είσοδο του χρήστη στο πρόγραμμα, επιτρέποντας την υποβολή ερωτημάτων και την αλληλεπίδραση με τη βάση γνώσης.

Παραδείγματα Χρήσης

1. **Εισαγωγή Βάσης Γνώσης**
   Ο χρήστης δημιουργεί ένα αρχείο κειμένου με το όνομα fol_testcase_1.txt. Το αρχείο περιέχει τις ακόλουθες προτάσεις:
    - Owns(John, Dog) ∨ ¬Loves(John, Dog)
    - Loves(John, Dog)

Το πρόγραμμα εισάγει τις προτάσεις αυτές στη βάση γνώσης και τις αποθηκεύει για μελλοντική χρήση.

2. **Υποβολή Ερωτήματος**
   Ο χρήστης εισάγει το ερώτημα:
   - Loves(John, Dog)

Το πρόγραμμα εξετάζει τη βάση γνώσης και επιστρέφει το αποτέλεσμα:

   - "Contradiction found: Goal is derivable."

3. **Περίπτωση Αντίφασης**
   Ο χρήστης εισάγει στη βάση γνώσης τις ακόλουθες προτάσεις:
   - Owns(John, Dog)
   - ¬Owns(John, Dog) ∨ Happy(John)

Στη συνέχεια, υποβάλλει το ερώτημα:

   - Happy(John)

Το πρόγραμμα επεξεργάζεται τις προτάσεις και επιστρέφει:

   - "Contradiction found: Goal is derivable."

Πειραματικά Αποτελέσματα

Δοκιμάστηκαν διαφορετικά σενάρια με βάσεις γνώσης και ερωτήματα, οδηγώντας στα εξής αποτελέσματα:

1. **Απλή βάση γνώσης**
   Βάση:
   - Owns(John, Dog)
   - ¬Owns(John, Dog) ∨ Happy(John)

Ερώτημα: Happy(John)
Αποτέλεσμα: Το πρόγραμμα εντόπισε αντίφαση, συνεπώς το ερώτημα αποδείχθηκε.

2. **Πολυπλοκότερη βάση γνώσης**
   Βάση:
   - Owns(John, Dog) ∨ Loves(John, Dog)
   - ¬Owns(John, Dog)

Ερώτημα: Loves(John, Dog)
Αποτέλεσμα: Το πρόγραμμα απέδειξε το ερώτημα συνδυάζοντας τις προτάσεις.

3. **Ασυνεπής βάση γνώσης**
   Βάση:
   - Owns(John, Dog)

- ¬Owns(John, Dog)

Οποιοδήποτε ερώτημα αποδεικνύεται λόγω της αντίφασης στη βάση γνώσης.

Συμπεράσματα

Η μέθοδος forward chaining υλοποιήθηκε με επιτυχία, παρέχοντας τη δυνατότητα εξαγωγής συμπερασμάτων από προτάσεις Horn. Η εφαρμογή υποστηρίζει σύνθετες βάσεις γνώσης και μπορεί να χρησιμοποιηθεί για τη διαχείριση λογικών κανόνων.

Propositional:

```
"C:\Program Files\Java\jdk-17\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\Intel
Processing line: !a V b
Processing line: !b V c V a
Processing line: !c V d
Processing line: !d
Type a siple literal aa query (q to exit): a
¬a v b , a v ¬b v c     ⊨     b v ¬b v c
¬a v b , a v ¬b v c     ⊨     a v ¬a v c
¬a v b , a     ⊨     b
a v ¬b v c , ¬c v d     ⊨     a v ¬b v d
¬c v d , ¬d     ⊨     ¬c
¬a v b , b v ¬b v c     ⊨     ¬a v b v c
¬a v b , a v ¬b v d     ⊨     b v ¬b v d
¬a v b , a v ¬b v d     ⊨     a v ¬a v d
a v ¬b v c , b     ⊨     a v c
a v ¬b v c , ¬c     ⊨     a v ¬b
b v ¬b v c , b     ⊨     b v c
b v ¬b v c , a v ¬b v d     ⊨     a v ¬b v c v d
b v ¬b v c , ¬c     ⊨     b v ¬b
a v ¬a v c , ¬c     ⊨     a v ¬a
b , a v ¬b v d     ⊨     a v d
¬a v b , b v ¬b v d     ⊨     ¬a v b v d
¬a v b , a v ¬b v c v d     ⊨     b v ¬b v c v d
¬a v b , a v ¬b v c v d     ⊨     a v ¬a v c v d
¬a v b , a v d     ⊨     b v d
a v ¬a v c , a v d     ⊨     a v c v d
¬a v b v c , b v ¬b v d     ⊨     ¬a v b v c v d
¬a v b v c , a v d     ⊨     b v c v d
New clauses were not found.
Result is false
Type a siple literal aa query (q to exit):
```

```
Type a siple literal aa query (q to exit): !a
¬a v b , a v ¬b v c     ⊨     b v ¬b v c
¬a v b , a v ¬b v c     ⊨     a v ¬a v c
¬a v b , a     ⊨     b
a v ¬b v c , ¬c v d     ⊨     a v ¬b v d
¬c v d , ¬d     ⊨     ¬c
¬a v b , b v ¬b v c     ⊨     ¬a v b v c
¬a v b , a v ¬b v d     ⊨     b v ¬b v d
¬a v b , a v ¬b v d     ⊨     a v ¬a v d
a v ¬b v c , b     ⊨     a v c
a v ¬b v c , ¬c     ⊨     a v ¬b
b v ¬b v c , b     ⊨     b v c
b v ¬b v c , a v ¬b v d     ⊨     a v ¬b v c v d
b v ¬b v c , ¬c     ⊨     b v ¬b
a v ¬a v c , ¬c     ⊨     a v ¬a
b , a v ¬b v d     ⊨     a v d
¬a v b , b v ¬b v d     ⊨     ¬a v b v d
¬a v b , a v ¬b v c v d     ⊨     b v ¬b v c v d
¬a v b , a v ¬b v c v d     ⊨     a v ¬a v c v d
¬a v b , a v d     ⊨     b v d
a v ¬a v c , a v d     ⊨     a v c v d
¬a v b v c , b v ¬b v d     ⊨     ¬a v b v c v d
¬a v b v c , a v d     ⊨     b v c v d
New clauses were not found.
Result is true
Type a siple literal aa query (q to exit):
```

First order:

```
"C:\Program Files\Java\jdk-17\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community
Processing line: ¬Loves(x, Animal) v Owns(x, Dog)
Processing line: ¬Kills(x, Animal) v ¬Loves(x, Animal)
Processing line: ¬Kills(John, Animal)
Processing line: Loves(x, Animal) v Kills(x, Animal)
Type a sentence aa query (q to exit): Loves(John,Dog)
Resolving [~Loves([x, Animal]), Owns([x, Dog])] with [Loves([x, Animal]), Kills([x, Animal])]
    New sentence: [Kills([x, Animal]), Owns([x, Dog])]
Resolving [~Kills([x, Animal]), ~Loves([x, Animal])] with [Loves([x, Animal]), Kills([x, Animal])]
    New sentence: [Loves([x, Animal]), ~Loves([x, Animal])]
Resolving [~Kills([John, Animal])] with [Loves([x, Animal]), Kills([x, Animal])]
    New sentence: [Loves([x, Animal])]
Resolving [Loves([x, Animal]), Kills([x, Animal])] with [~Kills([x, Animal]), ~Loves([x, Animal])]
    New sentence: [~Kills([x, Animal]), Kills([x, Animal])]
Resolving [Kills([x, Animal]), Owns([x, Dog])] with [~Kills([x, Animal]), ~Loves([x, Animal])]
    New sentence: [~Loves([x, Animal]), Owns([x, Dog])]
Resolving [Kills([x, Animal]), Owns([x, Dog])] with [~Kills([John, Animal])]
    New sentence: [Owns([x, Dog])]
Resolving [Loves([x, Animal]), ~Loves([x, Animal])] with [~Kills([x, Animal]), ~Loves([x, Animal])]
    New sentence: [~Kills([x, Animal])]
Resolving [Loves([x, Animal]), ~Loves([x, Animal])] with [Loves([x, Animal]), Kills([x, Animal])]
    New sentence: [Kills([x, Animal])]
Resolving [~Kills([x, Animal]), Kills([x, Animal])] with [~Kills([x, Animal]), ~Loves([x, Animal])]
    New sentence: [~Loves([x, Animal])]
Resolving [Kills([x, Animal])] with [~Kills([John, Animal])]Contradiction found: Goal is derivable.
```