Ηλίας Σέττας 3150156
Χρήστος Γκουρνέλος 3140033
Πέτρος Δημητρακόπουλος 3150034

Τεχνητή Νοημοσύνη Εργασία 2η: Λογική

Άσκηση 1

Για να φτιάξουμε ένα πρόγραμμα που κάνει **resolution** στην προτασιακή λογική, χρειαζόμαστε 3 κλάσεις και την Main, οι 3 κλάσεις αυτές είναι οι: Literal, CNFSubClause και CNFClause τις οποίες θα αναλύσουμε σύντομα. Εντελώς συνοπτικά, αυτό που κάνουμε με την resolution είναι όταν θέλω να αποδείξω κάτι, προσθέτω στη βάση γνώσης την άρνησή του και να κάνω resolution όλους τους όρους μεταξύ τους, καθώς και με τους νέους που παράγω με σκοπό να βγάλω το κενό σύνολο, πράγμα που θα με οδηγήσει σε άτοπο. Resolution μεταξύ 2 τύπων είναι να αφαιρώ 1 literal το οποίο έχει ίδιο όνομα αλλά αντίθετο πρόσημο, πχ Αν κάνω resolution στο (A∨ B) με το (¬A∨ ¬B) μπορεί να έχω ως αποτέλεσμα το B∨ ¬B ή το A∨ ¬A. Να σημειωθεί ότι παρόλο που πολλές φορές μπορεί να έχω πάνω από έναν τύπο που απλοποιείται (όπως στο παράδειγμα) θα αφαιρώ μόνο έναν κάθε φορά.

Literal:
Η κλάση αυτή αναπαριστά έναν τύπο στην προτασιακή λογική. Μπορούμε να θεωρήσουμε ότι ένας τύπος έχει 2 ορίσματα, το όνομά του (π.χ “P1) και το αν έχει άρνηση ή όχι. Έτσι η κλάση αυτή αποτελείται από αυτά τα 2 ορίσματα, getters-setters, μια συνάρτηση εκτύπωσης, συναρτήσεις για hashCode. Επίσης έχει και μια συνάρτηση σύγκρισης (compareTo) η οποία επιστρέφει 0 αν τα 2 literals έχουν ίδιο όνομα και ίδια άρνηση, -1 ή +1 αν έχουν ίδιο όνομα και διαφορετικό πρόσημο κοκ.

CNFSubClause:
Η κλάση αυτή αναπαριστά μία πρόταση στην προτασιακή λογική. Εφόσον έχουμε υποθέσει ότι οι προτάσεις μας είναι σε CNF, μπορούμε να πούμε ότι οι προτάσεις που διαβάζουμε – άρα και αναπαριστούμε – θα είναι αυτής της μορφής μόνο. Το CNFSubClause λοιπόν αναπαριστά μια πρόταση της μορφής (A∨ B∨ C ∨ ….). Προφανώς μπορούν να υπάρξουν και αρνήσεις. Για να γίνει η αναπαράσταση αυτή, χρησιμοποιούμε ένα HashSet το οποίο περιέχει Literals. Το hashSet μας βολεύει καθώς μας βοηθά να αποφύγουμε και την διαχείριση διπλών τύπων, οι οποίοι μπορούν απλά να παραληφθούν (πχ (A ∨ A∨ A) = A).Έτσι η κλάση περιέχει τους constructor της, getters setters και συναρτήσεις για σύγκριση. Αυτή η συνάρτηση που είναι σημαντικό να συζητήσουμε για αυτήν την κλάση είναι η **resolution(CNFSubClause SC1, CNFSubClause SC2).**Η συνάρτηση αυτή δέχεται ως όρισμα 2 CNFSubClauses τα οποία κάνει μεταξύ τους resolution. Πιο συγκεκριμένα αφού έχουμε πάρει τα iterators για τα 2 sub clauses μπαίνουμε σε μια λούπα η οποία τρέχει όσο ο

1ος όρος (SC1) έχει επόμενο όρο, για κάθε επανάληψη αυτής της πρώτης λούπας παίρνουμε την άρνηση του όρου στον οποίον δείχνει ο iterator, και βλέπουμε αν αυτός ο όρος υπάρχει στο SC2. Αν δεν υπάρχει απλώς συνεχίζουμε την επανάληψή μας. Αν όμως υπάρχει τότε δημιουργώ ένα νέο SubClause στο οποίο προσθέτω τα στοιχεία των SC1 και SC2 εκτός του στοιχείου το οποίο εξέτασα τώρα (και αυτού και της άρνησης του). Αυτό που θα παραχθεί το προσθέτω σε έναν Vector ο οποίος δέχεται sub clauses. Αυτός ο Vector είναι αυτό που επιστρέφει η συνάρτηση Resolution.

CNFClause:
Το CNFClause είναι μια κλάση που αναπαριστά μια πρόταση σε CNF. Είναι μια απλή κλάση που περιέχει απλώς ένα Vector το οποίο μέσα του έχει αντικείμενα CNFSubClause. Αυτή η κλάση θα χρησιμοποιηθεί για να αναπαραστήσει την βάση γνώσης μας. Αυτά που περιέχει ο πίνακας θα μπορούσαμε να πούμε ότι συνδέονται με ^ έτσι η έκφραση (Αν Β) ^ (Βν C) μπορεί να αναπαρασταθεί ως ένα αντικείμενο CNFClause το οποίο περιέχει 2 CNFSubClause, το Αν Β και το Βν C . Με την σειρά τους αυτά τα SubClause περιέχουν literals κοκ.

Main:
Όλη η βασική δουλειά του ερωτήματος γίνεται στην Main και κυρίως στην συνάρτηση PL_Resolution την οποία θα αναλύσουμε σε λίγο. Αρχικά το πρόγραμμα διαβάζει το Input από ένα αρχείο txt. Το input δίνεται σε μορφή μιας πρότασης CNF. Επίσης στο αρχείο εισόδου υπάρχει και η άρνηση της πρότασης που θέλουμε να αποδείξουμε. Αφού γίνουν απαραίτητα initialisations πινάκων, είμαστε έτοιμοι να τρέξουμε την PL_Resolution. Αρχικά η μέθοδος δημιουργεί ένα αντίγραφο της βάσης γνώσης μας, καθώς σε αυτό θα προσθέσουμε πράγματα και δεν θέλουμε το original να βγει παραμορφωμένο. Αφού προσθέσουμε και την άρνηση αυτού που θέλουμε να αποδείξουμε, είμαστε έτοιμοι να ξεκινήσουμε. Αρχικά υπάρχει ένας Vector ο οποίος λέγεται NEWsubclauses και θα περιέχει όλα τα νεα SubClauses που παράγονται. Αυτό που κάνουμε στη μέθοδο είναι να συγκρίνουμε το κάθε ένα από τα στοιχεία της βάσης γνώσης μας, δηλαδή το στοιχείο 0 με το 1, το 0 με το 2, με το 3 με το 4 κοκ. Έπειτα συγκρίνω το 1ο με το 2ο,με το 3ο,με το 4ο... κοκ έως ότου να έχω συγκρίνει όλα τα στοιχεία που έχει η βάση. Η σύγκριση που αναφέραμε είναι ότι ανάμεσα στα 2 στοιχεία Ci και Cj καλώ την συνάρτηση resolution της κλάσης CNFSubClause. Αν αυτό επιστρέψει το κενό σύνολο, δηλαδή αποτέλεσμα χωρίς όρους, σημαίνει ότι έχω αποδείξει αυτό που θέλω. Αν όμως επιστρέψει ένα αποτέλεσμα το οποίο περιέχει όρους , το προσθέτω στο NEWsubclauses. Όταν τελειώσω όλες τις συγκρίσεις, αυτό που κάνω είναι να προσθέσω τα στοιχεία του NEWsubclauses στο αντίγραφο της βάσης γνώσης μου. **ΠΡΟΣΟΧΗ : Αυτά που προσθέτω στη βάση γνώσης είναι <u>μόνο</u> καινούριοι τύποι, αν το NEWsubclauses <u>δεν</u> περιέχει κάποιον καινούριο τύπο (δηλαδή τύπο που να μην τον έχει ήδη η βάση γνώσης μου) τότε σταματάω την εκτέλεση καθώς δεν μπορεί να βρεθεί αυτό που θέλω να δείξω.**

Άσκηση 2

Πηγή για χρησιμοποιημένους τύπους/σύμβολα->https://en.wikipedia.org/wiki/Horn_clause

Η άσκηση 2 περιέχει αρκετά πράγματα που περιέχονται ήδη στην 1,έτσι για συντομία δεν θα τα αναπτύξουμε ξανά. Αρχικά η άσκηση 2 εκτελεί την απόδειξη μέσω της μεθόδου forward chaining στην προτασιακή λογική. Θεωρούμε δεδομένο ότι διαβάζει προτάσεις της μορφής

$$\neg p \vee \neg q \vee \ldots \vee \neg t \vee u$$

Όπου μπορούν να αναπαρασταθούν και ως

$$u \leftarrow p \wedge q \wedge \ldots \wedge t$$

Επίσης υπάρχει και ένας όρος, η agenda ο οποίος περιέχει τους τύπους τους οποίος γνωρίζω ότι είναι αποδεδειγμένοι αλλά δεν έχω εξετάσει ακόμα τις παρενέργειές τους στην βάση. Για την υλοποίηση της άσκησης 2, αρχικά χρησιμοποιήσαμε την κλάση **Literal** της προηγούμενης άσκησης χωρίς κάποια αλλαγή. Έπειτα χρησιμοποιήσαμε μια κλάση που ονομάζεται **CNFSubClause** η οποία είναι αρκετά όμοια με της προηγούμενης άσκησης αλλά έχει μερικές μικρές αλλαγές τις οποίες θα αναλύσουμε. Η κλάση αυτή δεν περιέχει 1 πίνακα με Literals όπως η προηγούμενη, αλλά 2 .Ο 1ος δείχνει ποια Literals περιέχονται στην πρόταση Horn Που αναπαριστά η κλάση και ο 2ος δείχνει ποιούς από τους όρους της πρότασης αυτής έχουμε αποδείξει. Επίσης υπάρχει και μια μεταβλητή τύπου Literal η οποία δείχνει τον τύπο που θέλουμε να αποδείξουμε,για παράδειγμα στην πρόταση $(p \wedge q \wedge \ldots \wedge t) \rightarrow u$ τα κομμάτια αριστερά του $\rightarrow$ περιέχονται στον 1ο πίνακα (literals) και το u στην μεταβλητή result. Επίσης υπάρχει και μια μεταβλητή που λέγεται proven και μας δείχνει αν έχει αποδειχτεί το result ¨η όχι (αναλόγως με το αν είναι true-false). Εκτός από τα κλασικά (getters-setters και συναρτήσεις εκτύπωσης-σύγκρισης) υπάρχει και μία συνάρτηση που λέγεται check_if_exists, αυτή η συνάρτηση ελέγχει αν το Literal που παίρνει ως όρισμα περιέχεται στον πίνακα Literals. Αν περιέχεται ελέγχει αν υπάρχει και στον πίνακα proven_literals. Αν υπάρχει δεν κάνει κάτι. Αν δεν υπάρχει προσθέτει το Literal και σε αυτόν τον πίνακα και έπειτα καλεί την check_proven. Η check_proven η οποία ελέγχει αν το μέγεθος του πίνακα που περιέχει τους όρους είναι ίσος σε μέγεθος με τον πίνακα που περιέχει τους αποδεδειγμένους όρους. Αν ναι τότε μπορεί με σιγουριά να πει ότι το result έχει γίνει proven. Εφόσον χρησιμοποιούμε hashset δεν υπάρχει περίπτωση να έχουμε έναν τύπο 2 φορές στο proven_things και επίσης ελέγχουμε ότι εισάγουμε εκεί να είναι ήδη στον πίνακα literals. Έτσι είμαστε σίγουροι ότι η συνάρτηση θα έχει πάντα σωστό αποτέλεσμα.

CNFClause:
Η κλάση αυτή είναι αρκετά αλλαγμένη από τις προηγούμενες και περιέχει και τη βασική συνάρτηση του ερωτήματος 2. Έτσι θα ήταν καλό να αναλυθεί ξεχωριστά. Αυτή η κλάση περιέχει έναν πίνακα με sub clauses. Επίσης περιέχει και έναν πίνακα ο οποίος δείχνει το ποιές μεταβλητές έχουν αποδειχθεί ότι υπάρχουν, καθώς και έναν πίνακα με literals ο οποίος λέγεται Agenda και θα αναλυθεί σύντομα. Μια συνάρτηση που αξίζει να αναλύσουμε πριν την forward_chaining είναι η iterate_list. Αρχικά αυτή η συνάρτηση δέχεται ως 1ο όρισμα έναν int. Αυτός ο int είναι ένας αριθμός ο οποίος ξεχωρίζει τις 2 διαφορετικές τις λειτουργίες.
Όταν ο i=1-> Ψάχνει στη βάση γνώσης μας για να βρει ποιοι από τους τύπους έχουν μόνο prove (**δηλαδή είναι γεγονότα**) όταν τους βρει αυτούς τους καταχωρεί ως proven και τους προσθέτει στην agenda.
Όταν ο i≠1-> Με όρισμα τον Literal που έχει ως 2ο όρισμα η συνάρτηση μας, πηγαίνουμε σε

κάθε τύπο της βάσης γνώσης και καλούμε την συνάρτηση check_if_exists με όρισμα τον τύπο που έχουμε δώσει ως όρισμα στην iterate_list. Αφού τελειώσει αυτή η λούπα κάνουμε πάλι προσπέλαση την βάση γνώσης ελέγχοντας αν υπάρχουν τύποι οι οποίοι είναι μεν proven, αλλά το αποτέλεσμά τους δεν περιέχεται στην Agenda. Αν βρω κάποιο τέτοιο sub clause, προσθέτω το result του στην agenda. Εφόσον έχουμε αναλύσει τα πάντα τόσο πολύ, η συνάρτηση Forward_Chaining είναι αρκετά απλή στην κατανόηση. Αρχικά δέχεται ως όρισμα μια βάση γνώσης και 1 Literal που θέλω να αποδείξω. Κάνει initialise την Agenda και καλεί την iterate_list με όρισμα i=1, το οποίο θα μας δώσει τους πρώτους τύπους που είναι αποδεδειγμένοι για να μπορέσουμε να κινηθούμε περαιτέρω. Έπειτα όσο η Agenda δεν είναι άδεια, παίρνω τον 1ο τύπο που περιέχει, κρατάω ένα αντίγραφό του και τον αφαιρώ από την Agenda.Αφού το κάνω αυτό ελέγχω αν ο τύπος αυτός περιέχεται στο proven_literals,αν περιέχεται πηγαίνω ξανά στην αρχή της λούπας και παίρνω τον επόμενο όρο της Agenda. Αν όμως δεν περιέχεται τότε τον προσθέτω στην λίστα proven_literals και καλώ την συνάρτηση iterate_list με ορίσματα (i=2 και τον Literal που πριν λίγο αφαίρεσα από την Agenda). Αυτό που θα γίνει έπειτα είναι να δει αν περιέχεται κάπου με σκοπό να κάνει trigger νέους κανόνες. Αν μετά από αυτό κατάφερα να αποδείξω έναν νέο τύπο ο οποίος είναι αυτός που θέλω να αποδείξω, επιστρέφω true και τελειώνω. Αν όμως δεν είναι ίδιος με κανέναν από αυτούς που περιέχονται στο proven_literals τότε συνεχίζω την επανάληψη, πάντα εφόσον έχω πράγματα στην Agenda.

Main:

Η main αρχικά διαβάζει τα στοιχεία από το αρχείο εισόδου. Τα στοιχεία είναι της ακόλουθης μορφής: κάθε γραμμή είναι και 1 horn clause, όλοι οι τύποι που πάνε αριστερά από το → είναι σε μορφή άρνησης. Ο μόνος τύπος που δεν έχει άρνηση είναι το result, το οποίο πάντα θα είναι στο τέλος της γραμμής. Αν κάποια γραμμή είναι της μορφής A, χωρίς τίποτα άλλο, τότε αυτό είναι ένα fact. Επίσης στην τελευταία γραμμή ως Prove γράφουμε τον τύπο τον οποίο θέλουμε να αποδείξουμε. Αφού διαβάσω τους τύπους της βάσης γνώσης μου και τον τύπο που θέλω να αποδείξω, καλώ τη μέθοδο forward chaining Και βλέπω το αποτέλεσμά της.

Άσκηση 4

Η 4η άσκηση μας ζητούσε να κάνουμε forward chaining σε First Order Logic. Θα αρχίσουμε λοιπόν να αναλύουμε το όλο πρόγραμμα και το σκεπτικό πίσω από αυτό κομμάτι-κομμάτι. Αρχικά, για να παραστήσουμε την πρωτοβάθμια κατηγορηματική λογική σε τζάβα, χρειαζόμαστε κλάσεις οι οποίες θα μπορέσουν να αναπαραστήσουν κάθε όρο ξεχωριστά. Οι τύποι των όρων μπορεί να είναι 4: α)Μεταβλητή β)Σταθερά γ)Συνάρτηση δ)Σχέση. Όμως, επειδή για συναρτήσεις σύγκρισης (ή πχ την Unify) οι οποίες θα μπορούσαν να δεχτούν οποιονδήποτε από τους 4 τύπους και στην περίπτωση της σχέσης θα μπορούσε να περιέχει πολλούς τύπους, θα μπλέκαμε τα πράγματα αρκετά. Έτσι δημιουργήσαμε ένα κοινό interface, το οποίο ονομάζεται **Unifiable.** Το Unifiable το κάνει implement οποιοδήποτε αντικείμενο μπορεί να γίνει Unify, δηλαδή κάθε κλάση που αναπαριστά ένα αντικείμενο στην FOL. Η Unifiable περιέχει 2 συναρτήσεις οι οποίες επιστρέφουν το όνομα και τον τύπο του αντικειμένου που εξετάζουμε και αυτό μας βοηθά για να κάνουμε τα casts, διότι για παράδειγμα δεν γίνεται να μετατρέψουμε ένα Constant σε Relationship. Αν το κάναμε αυτό το πρόγραμμα θα έκλεινε. Έτσι πριν κάνουμε κάποιο cast βεβαιωνόμαστε ότι το αντικείμενο που θα πειράξουμε, είναι σίγουρα του τύπου που πάμε να το κάνουμε. Επίσης υπάρχει μια συνάρτηση εκτύπωσης και μια συνάρτηση η οποία ονομάζεται appendString. Αυτή η συνάρτηση δέχεται ως όρισμα έναν int και πηγαίνει σε μια σχέση-συνάρτηση ή ακόμα και

μεταβλητή και κολλάει το νούμερο αυτό στο όνομα της μεταβλητής. Σε περίπτωση που έχουμε σχέση ή συνάρτηση πηγαίνει στην λίστα των ορισμάτων τους και πειράζει τις μεταβλητές. Μια άλλη συνάρτηση του unifiable είναι η isSame η οποία συγκρίνει 2 unifiable objects και μας λέει αν είναι ίδια ή όχι. Αυτό συνήθως το κάνει ελέγχοντας πάντα το όνομα και τον τύπο τους. Αν τα αντικείμενα που ελέγχουμε είναι μεταβλητές ή σταθερές, τότε είναι ίδια εφόσον έχουν αυτά τα 2 όμοια. Αν όμως ελέγχουμε μια σχέση ή μια συνάρτηση, αυτό δεν αρκεί (πχ η F(x) και η F(x,y)) δεν είναι ίδιες συναρτήσεις. Έτσι στην περίπτωση σχέσης-συνάρτησης ελέγχουμε τα ορίσματα τους 1 προς 1και αν είναι 1 προς 1 ίδια τότε μπορούμε με ασφάλεια να συμπεράνουμε ότι οι τύποι είναι ίδιοι. Να σημειωθεί ότι δεν έχουμε αναφέρει τίποτα για την Unify ακόμα. Θα γίνει εφόσον έχουμε εξετάσει όλες τις κλάσεις.

Ένας τύπος στην πρωτοβάθμια κατηγορηματική λογική θεωρείται μετονομασία κάποιου άλλου αν έχει μεταβλητές στις ίδιες θέσεις, ασχέτως από το όνομα και όλα τα άλλα τα έχει ίδια. πχ τα Mother(x) kai Mother(y) είναι το ένα μετονομασία του άλλου. Όμως το Mother(x,"Psita") Mother(y,"Mary") δεν είναι καθώς η σταθερά τους στη 2η θέση είναι

Όσον αφορά το α) για να αναπαραστήσουμε μια μεταβλητή δημιουργήσαμε την κλάση **Variable**. Η κλάση έχει 3 μεταβλητές String. Η μια ονομάζεται name και η άλλη base_name. Το name προφανώς είναι το όνομα της μεταβλητής, το base_name είναι το όνομα που είχε η μεταβλητή όταν δημιουργήθηκε (θα αναλύσουμε γιατί υπάρχει σύντομα). Το 3ο String που υπάρχει λέγεται type και δείχνει τον τύπο του αντικειμένου (δηλαδή για αντικείμενα Variable type = "Var") το type χρησιμοποιείται κυρίως για casting στις συναρτήσεις. Τέλος, η κλάση αυτή περιέχει συναρτήσεις ισότητας-getters-setters και την συνάρτηση unify και unify_var, τις οποίες θα αναλύσουμε προς το τέλος.

Όσον αφορά το β) για να αναπαραστήσουμε μια σταθερά, δημιουργήσαμε την κλάση **Constant.** Η κλάση αυτή έχει ως ορίσματα 2 String,το 1ο είναι το όνομα του αντικειμένου και το 2ο είναι ο τύπος του (type = "Con").Έχει επίσης και την συνάρτηση unify και τις κλασικές συναρτήσεις σύγκρισης,get-set.

Πριν πάμε να αναλύσουμε το γ και το δ καλό θα ήταν να αναλύσουμε πρώτα την κλάση **List**. Η κλάση αυτή αναπαριστά μια λίστα η οποία μπορεί να περιέχει μεταβλητές, σταθερές ή συναρτήσεις. Αυτή η κλάση περιέχει 2 μεταβλητές String, η 1η είναι το όνομα της λίστας και η 2η είναι ο τύπος της ("Lis"). Επίσης περιέχει και ένα ArrayList το οποίο έχει μέσα όλα τα ορίσματα της λίστας αυτής, τα ορίσματα υπάρχουν με την σειρά από αριστερά προς δεξιά στο (χ , y , z) το x είναι στη θέση 0,το y στην θέση 1 και το z στη θέση 2. Όλοι οι constructors της κάνουν προφανώς deep copy (εφόσον πρόκειται για αντιγραφή από άλλη λίστα-αντικείμενο). Οι συναρτήσεις που υπάρχουν είναι οι getters-setters, isSame που ελέγχει όπως είπαμε και πριν 1 προς 1 τα στοιχεία της λίστας για να βεβαιωθεί ότι είναι ίδια, μια συνάρτηση που προσθέτει στοιχεία στη λίστα (Ονομάζεται add και έχει ως όρισμα 1 Unifiable), εφόσον έχει βρει πρώτα τον τύπο τους και έχει δημιουργήσει ένα νέο αντικείμενο που είναι αντίγραφο αυτού που βάλαμε ως όρισμα. Υπάρχει και η συνάρτηση appendString που ψάχνει τη λίστα για μεταβλητές και τους προσθέτει τον int που έχει ως όρισμα, Άλλη μια συνάρτηση που υπάρχει είναι η applySubSet η οποία παίρνει ως όρισμα ένα SubstitutionSet. Αυτή η συνάρτηση διατρέχει όλη την λίστα και αν κάποιο στοιχείο της λίστας είναι Bound,το αντικαθιστά με την κατάλληλη τιμή. Να σημειωθεί ότι κάθε αντικατάσταση γίνεται δημιουργώντας καινούριο αντίγραφο, καθώς αν δεν γίνει έτσι θα μπλεχτούν τα Pointers και αν για παράδειγμα αλλάξουμε κάτι στοsubstitutionSet μπορεί να αλλάξει και στην λίστα μας, πράγμα που θα κάνει το πρόγραμμα να μη δουλέψει καλά. Προφανώς αν υπάρχουν

συναρτήσεις ως στοιχεία στη λίστα τότε η συνάρτηση θα διατρέξει και την λίστα αυτής της συνάρτησης για να κάνει τις αντικαταστάσεις που χρειάζονται. Η τελευταία συνάρτηση που θα αναλύσουμε είναι η isRenamed(List l).Αυτή η συνάρτηση δέχεται ως όρισμα μια λίστα και ελέγχει αν η λίστα του ορίσματος είναι μια μετονομασία της υπάρχουσας λίστας (αυτήν που αναφέρουμε με το this.list). Αν είναι μετονομασία επιστρέφουμε true, ειδάλλως false. Μας μένει και η συνάρτηση unify που θα αναλύσουμε μετά.

Όσον αφορά το γ) για να αναπαραστήσουμε μια συνάρτηση δημιουργήσαμε την κλάση **Function**. Η κλάση έχει ως μεταβλητές 2 String, 1 είναι το όνομά της συνάρτηση (πχ στην F(x) το όνομα είναι το F) και η άλλη είναι ο τύπος της (type=”Fun”). Επίσης έχουμε και ένα αντικείμενο List που παριστάνει τα ορίσματα της συνάρτησης. Όσον αφορά τις συναρτήσεις έχουμε μια συνάρτηση unify και μια συνάρτηση add Η οποία προσθέτει ένα deep copy του αντικείμενου που δίνουμε ως όρισμα στην λίστα της συνάρτησης. Η επόμενη συνάρτηση είναι η isRenamed που δέχεται ως όρισμα ένα αντικείμενο και το ένα αποτελεί rename του άλλου αν είναι ίδιου τύπου, έχουν ίδιο όνομα και στην περίπτωση της συνάρτησης η λίστα της μίας είναι μετονομασία της λίστας της άλλης. Έχουμε μια συνάρτηση appendString που καλεί την appendString της λίστας της συνάρτησης με όρισμα ένα int. Επίσης έχουμε μια συνάρτηση που λέγεται contains και δέχεται ως όρισμα ένα αντικείμενο, επιστρέφει true αν περιέχεται στη λίστα του και false αν δεν περιέχεται.

Όσον αφορά το δ) για να αναπαραστήσουμε μια σχέση δημιουργήσαμε την κλάση **Relationship.** Η αλήθεια είναι ότι η σχέση είναι σχεδόν ίδια με την συνάρτηση, αν προσέξει κανείς οι περισσότερες μέθοδοι τους είναι σχεδόν ίδιες με μόνη διαφορά το casting σε μερικούς τύπους. Βεβαίως η διαφορά σχέσης-συνάρτησης είναι στο τι γίνεται να περιέχει το κάθε ένα. Μία σχέση δεν γίνεται να περιέχει άλλη σχέση, αλλά περιέχει μόνο σταθερές, μεταβλητές και συναρτήσεις. Μια συνάρτηση γίνεται να περιέχει συναρτήσεις, σταθερές και μεταβλητές.

Εφόσον αναλύσαμε τώρα τα αντικείμενα Unifiable και μερικές βασικές τους ιδιότητες, πάμε να δούμε την συνάρτηση **Unify**.

Unification is a "pattern matching" procedure that takes two atomic sentences, called **literals**, as input, and returns "failure" if they do not match and a substitution list, Theta, if they do match. That is, unify(p,q) = Theta means subst(Theta, p) = subst(Theta, q) for two atomic sentences p and q.

- Theta is called the **most general unifier (mgu)**
- All variables in the given two literals are implicitly universally quantified
- To make literals match, replace (universally-quantified) variables by terms
- Unification algorithm

Να προστεθεί ότι το SubstitutionSet έχει και μια συνάρτηση appendString, η οποία κάνει αυτό στις άλλες κλάσεις, αλλά το εφαρμόζει στον πίνακα bindings. Επίσης μια μεταβλητή γίνεται να είναι Bound μόνο σε 1 πράγμα. δηλαδή αν έχω x/z1 και x/z2 στον ίδιο πίνακα κάτι δεν πάει καλά.

Η κλάση **Binding** αναπαριστά ένα Binding.

Η κλάση **Sentence** αναπαριστά μια πρόταση HORN η οποία αποτελείται από διάφορα

αντικείμενα Unifiable και έχει και 1 όρισμα prove. Έτσι ως όρισμα έχει μια ArrayList που αναπαριστά τα κομμάτια της πρότασης και μια μεταβλητή Unifiable prove η οποία αναπαριστά το συμπέρασμα, για παράδειγμα η πρόταση Loves(x,y) ^ Hates(x,z) => Hates(y,z) αναπαρίσταται ως ένα αντικείμενο Sentence έχοντας το κομμάτι αριστερά από το => στο ArrayList και το κομμάτι δεξιά από το => στο prove. Η κλάση έχει συναρτήσεις που προσθέτουν πράγματα στο ArrayList,που θέτουν το prove εκ' νέου, appendString για κάθε όρο της πρότασης,Unify ανάμεσα σε 2 προτάσεις, που εφόσον οι 2 προτάσεις έχουν το ίδιο μήκος, συγκρίνει τα ορίσματά τους. Να σημειωθεί ότι συγκρίνει τα ονόματα με ίδιο όνομα μεταξύ τους και αν τυχόν 1 στοιχείο της 1ης πρότασης δεν βρίσκει κάποιο στοιχείο στη 2η πρόταση με ίδιο όνομα, τότε επιστρέφεται null (δηλαδή fail). Τέλος υπάρχει και η συνάρτηση applySubSet η οποία πηγαίνει στο ArrayList και εφαρμόζει ένα SubstitutionSet.

Η κλάση στην οποία γίνονται τα περισσότερα πράγματα είναι η **KnowledgeBase.** Αρχικά έχει ως όρισμα ένα int limit το οποίο δείχνει τον max αριθμό επαναλήψεων π θα γίνουν στο forward_logic. Αυτό χρησιμοποιείται γιατί το forward_chaining είναι ημι-αποκρίσημο, άρα όταν δεν υπάρχει λύση δεν γνωρίζουμε αν θα καταφέρει να μας το πει. Η κλάση αυτή αναπαριστά μια βάση γνώσης. Πήραμε και χωρίσαμε τα στοιχεία της σε 2 μεγάλες κατηγορίες. τα στοιχεία που ανήκουν στην κάθε κατηγορία ανήκουν σε έναν ανάλογο πίνακα (ArrayList). Ο πίνακας sent περιέχει προτάσεις οι οποίες είναι <<κανόνες>>. Ο άλλος πίνακας (proven_things) περιέχει <<facts>>. Οι συναρτήσεις που πρέπει να αναφέρουμε πριν αναλύσουμε το forward chaining είναι οι εξής (τις οποίες λίγο πολύ τις έχουμε δει): new_vars(int a) η οποία δέχεται ένα όρισμα int και παίρνει αυτό το int και το εφαρμόζει σε όλες τις προτάσεις της βάσης γνώσης με την μέθοδο appendString(int a), την isRenamed(Unifiable a) που μας λέει αν το Unifiable a είναι μετονομασία οποιουδήποτε άλλου τύπου που περιέχεται στο proven_things και την getSubsets(ArrayList<Unifiable> superSet, int k) η οποία επιστρέφει έναν πίνακα με Sentences,όπου τα sentences σε αυτόν τον πίνακα είναι όλοι οι δυνατοί συνδυασμοί μήκους k που μπορεί να δημιουργηθούν με τα στοιχεία του πίνακα superSet.

Όλα όσα αναλύσαμε πιο πάνω χρειάζονται για να κατανοηθεί πλήρως το τι κάνει η μέθοδος **forward chaining**, την οποία θα περιγράψουμε τώρα.

Αρχικά δέχεται ως όρισμα 1 βάση γνώσης και 1 Unifiable, το οποίο και θέλουμε να αποδείξουμε. Η συνάρτηση ξεκινάει δημιουργώντας ένα αντίγραφο της βάσης γνώσης και 2 arrayLists, 1(ονομάζεται n) περιέχει Unifiable και το άλλο (ονομάζεται proven_n) Sentences. Έπειτα σε μια λούπα αρχικοποιούμε τους πίνακες και τρέχουμε την βάση γνώσης μας για να εκτελέσουμε την new_vars, αυτό μας αλλάζει τα ονόματα των τύπων και γίνεται για να μην υπάρξουν μπερδέματα με το substitutionSet, ο αριθμός που θα προστεθεί στα ονόματα καθορίζεται από την μεταβλητή count η οποία επίσης μετράει και τα συνολικά βήματα που έκανα. Έπειτα για κάθε κανόνα της βάσης γνώσης πηγαίνουμε και δημιουργούμε κάθε δυνατό συνδυασμό γεγονότων (οι συνδυασμοί έχουν μέγεθος ίδιο με το μέγεθος του κανόνα που εξετάζουμε) και αφού δημιουργήσουμε το δυναμοσύνολο αυτό, κάνουμε unify το κάθε 1 στοιχείο του δυναμοσυνόλου με τον συγκεκριμένο κανόνα. Αν το unify δεν βγάλει αποτυχία τότε στο αποτέλεσμα του κανόνα εφαρμόζουμε το substitutionSet που μόλις παρήχθει και αποθηκεύουμε το αποτέλεσμα (το οποίο είναι τύπου Unifiable και το ονομάζουμε b_). Αφού γίνει αυτό εξετάσουμε αν το b_ είναι μετονομασία κάποιου άλλου τύπου στην βάση γνώσης, αν είναι εξετάζω τον επόμενο τύπο διότι θα έχω εξετάσει ήδη κάποιο προηγούμενό του, αν δεν είναι τότε το προσθέτω στον πίνακα n και το κάνω unify με τον τύπο που θέλω να αποδείξω. Αν το αποτέλεσμα αυτού του unification δεν είναι αποτυχία τότε βρήκα την λύση και επιστρέφω αυτό το substitutionSet. Αν το αποτέλεσμα είναι αποτυχία. Έπειτα συνεχίζω και

προσθέτω όλα τα στοιχεία του n (δηλαδή όλα όσα δεν ήταν rename κάποιου άλλου γεγονότος) στα γεγονότα της βάσης γνώσης μου. Αν ο πίνακας n έχει παράξει έστω και 1 καινούριο γεγονός ξανά εκτελώ την λούπα από την αρχή. Οι μόνοι τρόποι για να σταματήσει ο αλγόριθμος είναι να αποδείξει αυτό που ήθελα, να σταματήσει να παράγει νέα στοιχεία ή να ξεπεράσει τα limit βήματα.

Το τελικό κομμάτι που πρέπει να αναλύσουμε είναι η main.

Η main διαβάζει τα στοιχεία εισόδου από ένα αρχείο txt. Τα στοιχεία για να μπορέσουμε να τα αναπαραστήσουμε εύκολα είναι της εξής μορφής :

R:Loves(V:x , F:G (V:x , C:Psita)) AND R:Hates (V:x) THEN R:Hates (V:x , C:Psita)

που δείχνει την πρόταση Loves(x,F(x,"Psita")) ^ Hates(x) =>Hates(x,"Psita")

Άρα πριν από κάθε όρο γράφουμε ένα γράμμα που δείχνει τον τύπο του.

Επίσης αν μια γραμμή είναι της μορφής R:Missile (C:M1) τότε σημαίνει ότι είναι γεγονός.

Ανάλογα με το αν είναι γεγονός ή όχι το στοιχείο που διαβάζουμε το προσθέτουμε στην ανάλογη θέση.

Σημειώστε ότι για κάθε «λέξη» απαιτείται κενό ώστε να γίνεται διαχωρισμός από τις υπόλοιπες. Πχ αν γράψετε R:Loves(V:x,V:y) αντί για R:Loves (V:x , V:y) θα σας πετάξει error.

Στην τελευταία θέση του αρχείου υπάρχει το εξής Prove: R:Criminal (C:West) το όποιο στη συγκεκριμένη περίπτωση μας δείχνει ότι θέλουμε να αποδείξουμε το Criminal("West")

Αφού διαβάσει το αρχείο τρέχει στη βάση γνώσης για να αποφασίσει αν αυτό που θέλουμε να αποδείξουμε ισχύει ή όχι και προφανώς στο τέλος επιστρέφει ανάλογο μήνυμα

Πηγές:

http://pages.cs.wisc.edu/~dyer/cs540/notes/fopc.html

https://en.wikipedia.org/wiki/Unification_(computer_science)#Substitution