Πανεπιστήμιο Θεσσαλίας
2013

Τίτλος

" Υποθετικός συλλογισμός με πλέγματα κανόνων λογικής προεπιλογής "

Title

" Hypothetical reasoning using lattice representations of default rules "

Διπλωματική Εργασία
της

Τάτση Μαρίας

Επιβλέποντες **:Δασκαλοπούλου Ασπασία**
Επίκουρη Καθηγήτρια Π.Θ.

Ακρίτας Αλκιβιάδης
Αναπληρωτής Καθηγητής Π.Θ.

Βόλος, Ιούνιος 2013

Ευχαριστίες

Στα πλαίσια της διπλωματικής εργασίας θα ήθελα να ευχαριστήσω την κυρία Δασκαλοπούλου Ασπασία - επίκουρη καθηγήτρια και επιβλέπουσα της πτυχιακής - για την καθοδήγησή της, τις πολύτιμες συμβουλές της και την στήριξή της.

Θα ήθελα να ευχαριστήσω επίσης τον κύριο Αλκιβιάδη Ακρίτα - αναπληρωτή καθηγητή και συνεπιβλέποντα της πτυχιακής μου.

Οφείλω ένα μεγάλο ευχαριστώ στην οικογένεια μου, για την αγάπη τους, την υπομονή τους και την στήριξή τους σε εμένα όλα αυτά τα χρόνια και ιδιαίτερα στον πατέρα μου και στον παππού μου.

Τέλος σημαντικό ρόλο κατείχαν οι φίλοι μου. Τους ευχαριστώ που είναι πάντα δίπλα μου και με στηρίζουν στα βήματα μου όλα αυτά τα χρόνια.

Βόλος, Ιούνιος 2013

Περίληψη

Σκοπός της παρούσας εργασίας είναι η υλοποίηση και ανάπτυξη της συλλογιστικής πορείας ενός πράκτορα, με γνώμονα την προεπιλογή.

Ένας πράκτορας λογισμικού πρέπει να είναι ευφυής, να αποφασίζει δηλαδή για τον εαυτό του αυτόνομα, τι μέτρα θα πάρει για να εκπληρωθούν οι στόχοι του. Πρέπει να είναι επίσης ορθολογικός, έτσι ώστε να μπορεί να επιλέξει την καλύτερη διαθέσιμη πορεία δράσης και να λειτουργεί σε ένα σύστημα ανοικτό, δυναμικό και μη-ντετερμινιστικό και να αλληλεπιδρά με αυτό (καθώς και με άλλους πράκτορες). Η αλληλεπίδραση αυτή καθορίζει τι ενέργειες μπορεί να ακολουθήσει, τι του επιτρέπεται και τι του απαγορεύεται.

Η προσέγγισή μας βασίζεται στη σύνταξη κανόνων λογικής προεπιλογής (DFL) (Reiter 1980). Παρουσιάζουμε μία υπολογιστική τεχνική για τον τρόπο λήψης αποφάσεων ενός πράκτορα. Ο πράκτορας, ανάλογα με την τρέχουσα γνώση σε κάθε δεδομένη χρονική στιγμή, εντοπίζει τις κατάλληλες υποθέσεις και δρα σύμφωνα με αυτές. Όταν οι επιλογές πρέπει να αναθεωρηθούν, οι πράκτορες μπορούν να ανακατασκευάσουν την άποψή τους για τον περιβάλλον στο οποίο βρίσκονται και να δημιουργήσουν αναθεωρημένες απαιτήσεις και υποθέσεις.

Στόχος μας λοιπόν είναι για έναν πράκτορα ο οποίος δρα βάσει λογικής με προεπιλογή να εξάγει συμπεράσματα ακόμα και όταν δεν έχει γνώση για το περιβάλλον του. Αυτό γίνεται μέσω λήψης υποθέσεων.

Abstract

In the present paper we discuss the implementation and development of an agent reasoning scheme which is based on Default Logic.

An agent should be intelligent; he should be able to act on himself in order to accomplish his goals. He must also be rational, in the sense that he chooses the best available course of action and can function in an open, dynamic and non deterministic system. He should be able to interact with both the system as well as the additional agents that may be present in it.

Our approach is based on Default Logic Rules (DFL) (Reiter 1980). We present a computational technique to address the problem of deciding the course of action which should be taken by an agent in order to reach a conclusion. An agent according to its current knowledge can, at any given time, reach the appropriate assumptions and act accordingly, in order to draw conclusions. When these are no longer applicable, agents can reconstruct their view of the environment, so that they can continue reaching conclusions.

Therefore, when an agent acts based on Default Logic, we intent to draw conclusions even if he has insufficient knowledge of his environment which is accomplished by making assumptions.

Πίνακας Περιεχομένων

1

Εισαγωγή

1.1 Αντικείμενο Διπλωματικής

Δεν είναι λίγες οι φορές όπου στην καθημερινότητά μας κάνουμε υποθέσεις για να επιτευχθεί μία πρόβλεψη, ελπίζοντας να βεβαιωθεί-εξακριβωθεί η αλήθεια.

Ένα έξυπνο σύστημα, κάνοντας μία διαδικασία συλλογισμού, έχει ως στόχο να εξάγει σωστά αποτελέσματα. Υπάρχουν περιπτώσεις όμως, όπου το σύστημα έχει ελλιπείς πληροφορίες, είτε γιατί δεν είναι διαθέσιμες, είτε γιατί πρέπει να αντιδρά γρήγορα, μη έχοντας τον απαραίτητο χρόνο να συλλέξει όλα τα απαιτούμενα δεδομένα Ο υποθετικός συλλογισμός δημιουργήθηκε για το σκοπό αυτό. Χρησιμοποιεί κατασκευασμένες καταστάσεις-ιδέες και μέσω της λογικής και των πληροφοριών εξάγει συμπεράσματα. Στηρίζεται στην ιδέα ότι ένα γεγονός θα πραγματοποιηθεί εφόσον κάποιο ή κάποια άλλα έχουν ήδη επιβεβαιωθεί. Με τη χρήση υποθέσεων ένα πρόγραμμα ξεφεύγει από τη στερεότυπη απάντηση "σωστό ή λάθος" και παρέχει τη δυνατότητα μιας πιο διευρυμένης απάντησης, αυτής που θα στηρίζεται σε υποθέσεις.

Βασιζόμενοι στον υποθετικό συλλογισμό, η εργασία αυτή έχει ως στόχο την ανάπτυξη ενός συστήματος λήψεων αποφάσεων κάνοντας χρήση κανόνων λογικής προεπιλογής. Ένας από τους πιο διαδεδομένους συλλογισμούς είναι η Default Logic (λογική προεπιλογής), μέσω της οποίας μπορούν να εκφραστούν γεγονότα που στηρίζονται σε υποθέσεις και όχι μόνο γεγονότα που είναι ρητά αληθή ή ψευδή .

Κάνοντας χρήση της γλώσσας *Prolog* , δημιουργήσαμε μία βιβλιοθήκη (assumptions.pl), μέσω της οποίας προσφέρουμε την δυνατότητα υποστήριξης υποθέσεων σε γραμματική τύπου Default Logic. Παρέχουμε έτσι τη δυνατότητα να απαντηθεί μία ερώτηση κάνοντας υποθέσεις, εκτός από το πρότυπο της κλασικής λογικής, σύμφωνα με το οποίο παίρνουμε ρητή απάντηση (κατάφαση ή άρνηση).

Η εργασία αυτή έχει ως στόχο να δείξει μία πιο διευρυμένη λειτουργία των πρακτόρων λογικής. Η διαφοροποίηση μας με την κλασική λογική έγκειται στο γεγονός ότι ενώ ένας πράκτορας βασισμένος στη γνώση περιγράφεται σε τρία επίπεδα [5] :

- **Επίπεδο Γνώσης (knowledge level) :** σε αυτό το επίπεδο ο πράκτορας προσδιορίζεται λέγοντας τι γνωρίζει για τον κόσμο και ποιοι είναι οι στόχοι του.
- **Λογικό Επίπεδο (logical level) :** το επίπεδο όπου η γνώση και οι στόχοι του πράκτορα κωδικοποιούνται σε προτάσεις λογικής γλώσσας.
- **Επίπεδο Υλοποίησης (implementation level) :** το επίπεδο στο οποίο οι προτάσεις υλοποιούνται από ένα πρόγραμμα που τρέχει πάνω στην αρχιτεκτονική του πράκτορα

ο δικός μας πράκτορας εμφανίζει και ένα τέταρτο επίπεδο :

- **Επίπεδο Υποθέσεων (assumptions level) :** το επίπεδο όπου παρέχεται η δυνατότητα στον πράκτορα να εξάγει συμπέρασμα κάνοντας υποθέσεις.

Με τη χρήση των υποθέσεων θεωρούμε ότι ο εκάστοτε πράκτορας θα αποκτήσει περισσότερες δυνατότητες λήψης αποφάσεων. Έτσι, μέσω αλυσιδωτών υποθέσεων, μπορεί πλέον να καταλήξει σε μεγαλύτερο εύρος συμπερασμάτων, πράγμα το οποίο και επιθυμούμε.

Δημιουργώντας λοιπόν τη βιβλιοθήκη "assumptions.pl" και εφαρμόζοντας σε αυτή ένα πρόγραμμα συναλλαγής μεταξύ ενός αγοραστή και ενός πωλητή (το οποίο αναλύεται στο κεφάλαιο 5), θα διαπιστώσουμε τη δυνατότητα καταφατικών, αρνητικών και πιθανολογικών απαντήσεων.

1.2 Εφαρμογές της Default Logic

Η Default Logic μπορεί να εφαρμοστεί σε συστήματα που αναλύουν τη μορφολογία, τη γραμματική και την σημασιολογία λέξεων/φράσεων [12,13]. Γενικά αυτού του είδους η ανάλυση είναι απαραίτητη για συστήματα που διαχειρίζονται τον ανθρώπινο γραπτό λόγο. Μία περίπτωση τέτοιων συστημάτων είναι οι μηχανές ανάκλησης πληροφορίας.

Επίσης η Default Logic εμφανίζεται και στην δημιουργία αυτόματων συστημάτων εκμάθησης. Μέσω αυτής τέτοιες πλατφόρμες μπορούν να βαθμολογήσουν απαντήσεις οι οποίες είναι είτε ελλιπείς είτε λανθασμένα διατυπωμένες.

Τέλος η Default Logic συναντάται και στην θεωρία παιγνίων [14]. Για παράδειγμα η ισορροπία Nash, μπορεί να μοντελοποιηθεί μέσω γραμματικής τύπου Default Logic. Δύο ή περισσότεροι πράκτορες βρίσκονται σε ισορροπία, όταν με τις αποφάσεις που παίρνουν οδηγούνται σε καταστάσεις που έχουν τη μεγαλύτερη χρησιμότητα για όλους.

1.3 Οργάνωση Κειμένου

Η διάρθρωση του κειμένου της διπλωματικής εργασίας ακολουθεί την παρακάτω πορεία: στο κεφάλαιο 1 γίνεται μία εισαγωγή στο θέμα της διπλωματικής εργασίας καθώς αναφέρεται και η χρησιμότητά της στην καθημερινότητα μέσω καθημερινών εφαρμογών. Στο κεφάλαιο 2 παρουσιάζουμε το θεωρητικό υπόβαθρο, όπου και παραθέτουμε τις γνώσεις μας πάνω στις ανάγκες που απαιτήθηκαν για την εργασία μας. Στο κεφάλαιο 3 γίνεται η γνωριμία και η ανάλυση του αντικειμένου μας και των κανόνων του (Default Logic). Στο κεφάλαιο 4 προχωρούμε στο σχεδιασμό και στην υλοποίηση του προγράμματος μας. Στο κεφάλαιο 5 αποτυπώνουμε τα συμπεράσματά μας και προτείνουμε μελλοντικές εργασίες. Εν συνεχεία ακολουθούν τα παραρτήματα, όπου στο Α δίνουμε οδηγίες χρήσης της πλατφόρμας swi-Prolog, στο Β τον κώδικα της βιβλιοθήκης των υποθέσεων και στο Γ το σενάριο του τρεξίματος μας (σενάριο buyer-seller) και τον κώδικα του προγράμματος το οποίο ελέγχουμε. Τέλος παραθέτουμε τη βιβλιογραφία που απαιτήθηκε για να υλοποιηθεί η εργασία μας.

2

Θεωρητικό Υπόβαθρο

Για τις ανάγκες της παρούσας εργασίας, χρειάστηκε αρχικά να έρθουμε σε επαφή με τις έννοιες του πράκτορα και των πολυπρακτορικών συστημάτων, να αντιληφθούμε δηλαδή τον τρόπο σκέψης και δράσης ενός πράκτορα και την αλληλεπίδραση του με το περιβάλλον του.

Δεδομένου της ανάγκης για συγγραφή κώδικα, γνωρίσαμε τη γλώσσα *Prolog*, με τις λειτουργίες και τους κανόνες της. Ψάχνοντας έπειτα ένα περιβάλλον εκτέλεσης, ήρθαμε σε επαφή με την πλατφόρμα SWI-Prolog, μέσω της οποίας υλοποιήθηκε το πρόγραμμά μας. Μας χρειάστηκε επίσης βασική γνώση των βάσεων δεδομένων και ως επί το πλείστον σχεσιακών βάσεων δεδομένων .

Τέλος, στο πλαίσιο της υλοποίησης, αποκτήθηκε πλήρης κατανόηση του συλλογισμού με τη χρήση προεπιλογών και πιο συγκεκριμένα της Default Logic. Αντιληφθήκαμε τους κανόνες αυτής της λογικής και προβήκαμε στη δημιουργία μιας βιβλιοθήκης (" assumptions.pl "), μέσω της οποίας παρέχεται η δυνατότητα του υποθετικού μας συλλογισμού.

Στο κεφάλαιο που ακολουθεί αναλύονται βασικές έννοιες, απαραίτητες για την κατανόηση της συνολικής εργασίας μας.

2.1 Η έννοια του πράκτορα λογισμικού

Σύμφωνα με τους ***Wooldridge και Jennings*** [10] πράκτορας είναι οτιδήποτε μπορεί να αντιληφθεί το περιβάλλον του μέσω αισθητήρων και να αντιδράσει στο περιβάλλον μέσω μηχανισμών δράσης. Ένα σύστημα υλικού ή λογισμικού με τις ακόλουθες ιδιότητες:

αυτονομία: είναι ικανοί να αποφασίσουν από μόνοι τους τι ενέργεια πρέπει να κάνουν έτσι ώστε να πραγματοποιήσουν τους στόχους τους.

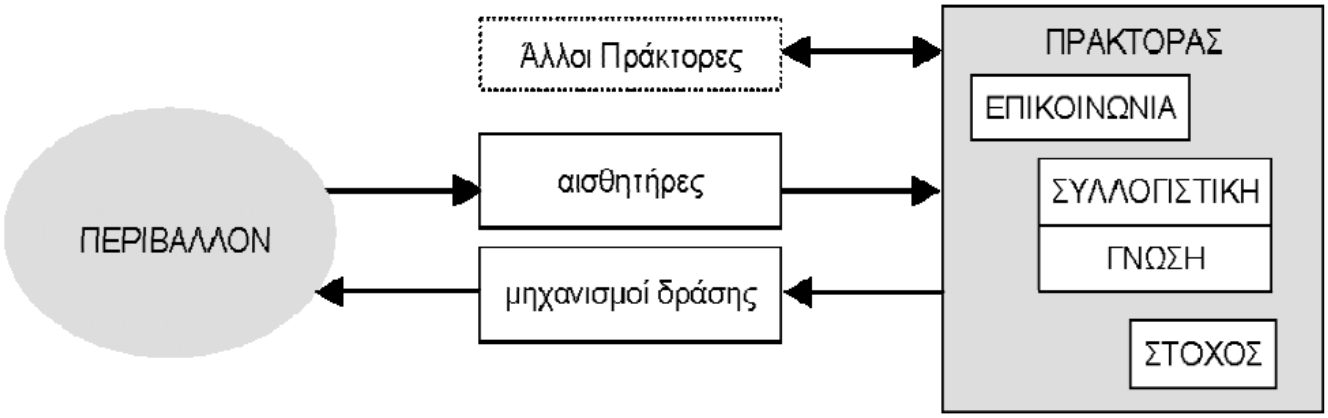
κοινωνικότητα: καθώς είναι σε θέση να δέχονται ερεθίσματα από το περιβάλλον τους και από άλλους πράκτορες και να αλληλεπιδρούν με αυτά.

Στον δικό μας πράκτορα, καθώς και σε όποιον παρέχεται δυνατότητα συλλογισμού, εμφανίζονται και άλλα δύο χαρακτηριστικά:

αντιδραστικότητα: αντιλαμβάνονται το περιβάλλον τους (το οποίο μπορεί να είναι ένας φυσικός κόσμος, μία συλλογή από άλλους πράκτορες, το διαδίκτυο ή και όλα μαζί συνδεδεμένα) και ανταποκρίνεται εγκαίρως στις αλλαγές που συμβαίνουν σε αυτό.

προνοητικότητα: δεν ενεργεί απλά ως απάντηση στο περιβάλλον του, αλλά είναι σε θέση να παρουσιάζει στόχους λαμβάνοντας πρωτοβουλία.

Στην εικόνα που ακολουθεί παρατηρούμε την αλληλεπίδραση ενός πράκτορα με το περιβάλλον του ή και με άλλους πράκτορες. Έχοντας μία βάση γνώσης και δυνατότητα συλλογισμού, δέχεται ερεθίσματα από το περιβάλλον του και μέσω των μηχανισμών δράσης πραγματοποιεί τον εκάστοτε στόχο του.



2.2 Η γλώσσα Prolog

Η Prolog[(1)] είναι μια δηλωτική-λογική γλώσσα προγραμματισμού, η οποία σχετίζεται με την τεχνητή νοημοσύνη. Στην Prolog, δεν μπορεί ο χρήστης να γράψει ό,τι ο υπολογιστής πρέπει να

κάνει γραμμή προς γραμμή, πράγμα που γίνεται σε διαδικαστικές γλώσσες όπως η C και η Java. Η γενική ιδέα πίσω από δηλωτικές γλώσσες είναι ότι θα περιγράψει την παρούσα κατάσταση (database and clauses). Το λογικό πρόγραμμα εκφράζεται από την άποψη των σχέσεων και ένας υπολογισμός θα ξεκινήσει εκτελώντας ένα ερώτημα (query), το οποίο θα εκτελεστεί μέσω αυτών των σχέσεων.

Με βάση αυτόν τον κώδικα, ο διερμηνέας ή μεταφραστής θα εξάγει λύση γνωστοποιώντας:

- αν μια πρόταση Prolog είναι αλήθεια ή όχι
- εάν περιέχει μεταβλητές
- ποιες είναι οι τιμές των μεταβλητών που πρέπει να βρίσκονται

2.2.1 Τύποι δεδομένων στην Prolog

Τα δεδομένα που χρησιμοποιούνται στην Prolog μπορούν να είναι της μορφής [4]:

- **Κατηγορήματα:** γενικού σκοπού, χωρίς κάποια εγγενή έννοια. Επί παραδείγματι, κατηγορήματα θεωρούνται τα (x,y,blue,"Skip")
- **Αριθμοί:** ακέραιοι ή και δεκαδικοί
- **Μεταβλητές:** συμβολίζονται από string γραμμάτων, αριθμών ή χαρακτήρων υπογράμμισης. Αρχίζουν με κεφαλαίο γράμμα, ή με κάτω-παύλα(δήλωση οποιασδήποτε μεταβλητής). Οι μεταβλητές χρησιμοποιούνται ως χαρακτήρες κράτησης θέσης για αυθαίρετους όρους. Ως ανώνυμη μεταβλητή χαρακτηρίζεται η '_' και ερμηνεύεται ως "κάποιο" αντικείμενο
- **Σύνθετοι όροι:** γράφονται ως "functor", ακολουθούμενοι από μία λίστα όρων, διαχωρισμένη με κόμματα. Σύνθετος όρος θεωρούνται τα: ("Maria", 1821), (Zelda[tom,jim])
- **Λίστες:** η σημαντικότερη δομή δεδομένων στην Prolog. Είναι ένας μονοδιάστατος πίνακας με στοιχεία του αριθμούς, σύμβολα, σύνθετους όρους ή και άλλες λίστες. Παραδείγματα λίστας: [physics, 8 , maths, 9] , [nick,mike,john], [3,nick,point(1,2), [a,b]] και η κενή λίστα [].
- **Strings:** ακολουθία χαρακτήρων που περιβάλλονται από εισαγωγικά.

2.2.2 Κανόνες και γεγονότα στην Prolog

Όπως αναφέρθηκε η Prolog περιγράφει σχέσεις. Οι σχέσεις αυτές χωρίζονται σε κανόνες και γεγονότα. Ένας κανόνας θα ισχύει εφόσον το αριστερό μέρος του είναι αληθές, κατά συνέπεια θα είναι και το δεξί του μέρος αληθές. Για παράδειγμα από τον κανόνα

Head :- Body.

αντιλαμβανόμαστε ότι "Head is true if_f Body is true", δηλαδή το αριστερό μέρος του κανόνα θα είναι σωστό, αν και μόνο αν το δεξί μέρος είναι σωστό. Το σώμα του κανόνα αποτελείται από κλήσεις σε κατηγορήματα. Θα μπορούσαμε δηλαδή να έχουμε στο δεξί μέλος και άλλα κατηγορήματα, διαχωρισμένα με κόμμα, τα οποία θα έπρεπε να ισχύουν όλα για να ισχύσει ο κανόνας μας.
Οι δηλώσεις με κενό σώμα (δεξί μέλος) ονομάζονται γεγονότα, καθώς ισχύουν πάντα και δεν χρειάζονται απόδειξη για την ισχύ τους. Παράδειγμα κανόνα είναι το:

human(tom). Από αυτή τη δήλωση αντιλαμβανόμαστε ότι ο tom είναι άνθρωπος.

Ο παραπάνω κανόνας είναι ισοδύναμος με το:

human(tom):- true, όπου το ενσωματωμένο κατηγόρημα true/0 είναι πάντα αληθές.

2.2.3 Ερωτήσεις στην Prolog

Έχοντας το παραπάνω γεγονός **human(tom).** Η ερώτηση που θα θέταμε είναι της μορφής

?-human(tom).
True

?-human(X).
X=tom
Μας επιστρέφει την τιμή(όνομα του ανθρώπου)

Αν προσθέταμε έναν κανόνα, έστω:

mammal(X):-human(X).

και ρωτούσαμε:

?-mammal(tom).
True

?- mammal(jim).
False

?-mammal(X).
X=tom

2.2.4 Μεταβλητές στην Prolog

Οι μεταβλητές [3] είναι συμβολοσειρές που μπορεί να περιέχουν γράμματα, ψηφία ή τον χαρακτήρα «_», και πρέπει πάντα να ξεκινούν με κεφαλαίο γράμμα ή με τον χαρακτήρα «_», π.χ.

X, Result, Object2, _a23

Οι μεταβλητές που ξεκινούν με τον χαρακτήρα «_» ονομάζονται ανώνυμες μεταβλητές. Χρησιμοποιούνται στις περιπτώσεις που δεν μας ενδιαφέρει η τιμή που θα λάβει η συγκεκριμένη μεταβλητή, π.χ.

?- father(peter,_).

Στην Prolog, παρατηρούμε ότι σε μία μεταβλητή η οποία έχει πάρει μία τιμή δεν μπορεί να δοθεί νέα (non destructive assignment). Τέλος οι μεταβλητές δεν έχουν τύπο (typeless) και άρα δεν απαιτούν δηλώσεις. Μπορούν να πάρουν ως τιμή οποιονδήποτε όρο.

[(1)]Το όνομα Prolog το έβγαλε ο συνεργάτης του Kowalski' Philippe Roussel και είναι συντομογραφία του γαλλικού «PROgramation et LOGique» (*«Προγραμματισμός και Λογική»*)

2.2.5 Αναδρομικοί Κανόνες

Ένας κανόνας στην Prolog που περιλαμβάνει στο σώμα του κάποια κλήση στο κατηγόρημα της κεφαλής του κανόνα καλείται αναδρομικός [3].

Παράδειγμα, ο ορισμός προγόνου:
Ο Α είναι πρόγονος του Β αν
ο Α είναι γονέας του Β ή αν
υπάρχει κάποιος C που ο Α είναι γονέας του και ο C είναι πρόγονος του Β.

```
ancestor(X,Y):- father(X,Y).
ancestor(X,Y):- father(X,Z), ancestor(Z,Y).
```

Ο πρώτος από τους δύο παραπάνω κανόνες ονομάζεται τερματική συνθήκη του αναδρομικού κανόνα και ο δεύτερος κανόνας μας δηλώνει την αναδρομή, αφού μέσα στο "Body" του (αριστερό μέρος της συνθήκης), καλείται ξανά το "Head" (is_ancestor).

Η αναδρομή διαδραματίζει σημαντικό ρόλο στη Prolog, καθώς χωρίς αυτή, θα έπρεπε να ορίσουμε πολλούς μη αναδρομικούς κανόνες της μορφής:

```
ancestor(X,Y):- father(X,Y).
ancestor(X,Y):- father(X,Z),father(Z,Y).
.....
ancestor(X,Y):- father(X,Z1),father(Z1,Z2),.....,father(ZN,Y).
```

Πράγμα χρονοβόρο, κουραστικό και ανώφελο.

2.3 Βάσεις Δεδομένων

Μια βάση δεδομένων [4] είναι ένας οργανωμένος τρόπος αποθήκευσης πληροφοριών και πρόσβασής τους με πολλούς τρόπους από διάφορα προγράμματα. Οι στόχοι μιας βάσης δεδομένων σύμφωνα με τον ***Δρ. Θεοδώρου Παύλο*** [11] είναι :

- ο περιορισμός της πολλαπλής αποθήκευσης των ίδιων στοιχείων (redundancy).
- ο καταμερισμός (sharing) των ίδιων στοιχείων σε όλους τους χρήστες.
- η ομοιομορφία (uniformity) στον χειρισμό και την αναπαράσταση των δεδομένων.
- η διατήρηση της ακεραιότητας (integrity) και της αξιοπιστίας (reliability) των δεδομένων.
- η ανεξαρτησία των δεδομένων (data independence) και των προγραμμάτων από τον φυσικό τρόπο αποθήκευσης των δεδομένων.

Κατά την υλοποίηση της βιβλιοθήκης μας κάναμε χρήση των σχεσιακών βάσεων δεδομένων. Με τον όρο σχεσιακή βάση δεδομένων εννοείται μία συλλογή δεδομένων οργανωμένη σε συσχετισμένους πίνακες που παρέχει ταυτόχρονα ένα μηχανισμό για ανάγνωση, εγγραφή, τροποποίηση ή και πιο πολύπλοκες διαδικασίες πάνω στα δεδομένα. Ο σκοπός μιας βάσης δεδομένων είναι η οργανωμένη αποθήκευση πληροφορίας και η δυνατότητα εξαγωγής της πληροφορίας αυτής, ιδίως σε πιο οργανωμένη μορφή, σύμφωνα με ερωτήματα που τίθενται στη σχεσιακή βάση δεδομένων. Τα δεδομένα είναι δυνατόν να αναδιοργανώνονται με πολλούς διαφορετικούς τρόπους, σε νοητούς πίνακες, χωρίς να είναι απαραίτητη η αναδιοργάνωση των φυσικών πινάκων που τα αποθηκεύουν. Στο σχεσιακό μοντέλο παρέχεται η δυνατότητα συσχέτισης

των πινάκων μέσω διακριτών πεδίων.

Για το συσχετισμό δύο πινάκων αρκεί ένα κοινό πεδίο, χαρακτηριστικό που κάνει το μοντέλο εύκαμπτο.

Επιλέξαμε τη σχεσιακή βάση δεδομένων γιατί a) είναι πιο ευέλικτη και b)ανταποκρίνεται σε όλες τις ερωτήσεις (μαθηματικές εντολές, πράξεις Boolean).

2.3.1 Χαρακτηριστικά Πίνακα Εγγραφών

Η είσοδος παρουσιάζει ένα στοιχείο και δεν υπάρχουν επαναλαμβανόμενες ομάδες στοιχείων. Σε κάθε στήλη όλα τα πεδία είναι του ίδιου είδους. Κάθε στήλη παίρνει δικό της όνομα. Όλες οι σειρές είναι διακριτές και δεν επιτρέπονται διπλές σειρές. Τόσο οι στήλες όσο και οι σειρές μπορούν να αντιμετωπιστούν με οιανδήποτε σειρά, οιανδήποτε στιγμή, χωρίς να επηρεαστεί ούτε το πληροφοριακό περιεχόμενο ούτε η φύση των λειτουργιών (σχέσεων).

2.4 Πλατφόρμα SWI-Prolog

Η SWI-Prolog είναι μια εφαρμογή ανοιχτού κώδικα της γλώσσας προγραμματισμού Prolog. Διαθέτει ένα πλούσιο σύνολο γραφικών, βιβλιοθηκών και πολλά πακέτα interface, για multithreading, unit testing, GUI. Η SWI-Prolog υποστηρίζει Unix, Windows, Macintosh και Linux πλατφόρμες.

2.5 Default Logic

Η Default logic είναι μη-μονοτονική λογική[(2)] που προτάθηκε από τον Raymond Reiter για να υποστηρίξει συλλογιστική με default υποθέσεις.

Η Default logic μπορεί να εκφράσει γεγονότα όπως “by default, κάτι είναι αλήθεια”, εν αντίθεση η standard logic μπορεί να εκφράσει μόνο κάτι αληθές ή ψευδές. Αυτό είναι πρόβλημα διότι ο συλλογισμός συχνά περιλαμβάνει γεγονότα τα οποία είναι αληθή στην πλειοψηφία τους, αλλά όχι πάντοτε. Ένα κλασσικό παράδειγμα είναι: “τα πουλιά τυπικά πετάνε”.Αυτός ο κανόνας μπορεί να εκφραστεί στην standard logic είτε ως “όλα τα πουλιά πετούν”, το οποίο δεν συνάδει με το γεγονός ότι οι πιγκουίνοι δεν πετούν ή ότι “όλα τα πουλιά που δεν είναι πιγκουίνοι και δεν είναι στρουθοκάμηλοι και δεν είναι...πετούν”, το οποίο υποχρεώνει όλες οι εξαιρέσεις του κανόνα να οριστούν. Η Default logic έχει ως στόχο να υποστηρίξει κανόνες σαν και αυτόν χωρίς να γίνει ειδική αναφορά όλων των εξαιρέσεων.

(2)μη μονοτονική λογική: είναι μία λογική, όπου η σχέση των συνεπειών είναι μη μονοτονική. Οι περισσότερες λογικές έχουν μονοτονική σχέση των συνεπειών τους, πράγμα που σημαίνει ότι προσθέτοντας έναν κανόνα στη θεωρία, ποτέ δεν παράγεται μειωμένο το σύνολο των συνεπειών. Μαθαίνοντας δηλαδή, ότι ένας καινούριος κανόνας ισχύει, δεν μειώνεται το σύνολο της βάσης γνώσης μας, σε αντίθεση με τη μη μονοτονική λογική[4].

3

Default Logic

3.1 Default Logic-γνωριμία

Όταν ένα έξυπνο σύστημα (είτε υπολογιστής, είτε άνθρωπος) προσπαθεί να λύσει ένα πρόβλημα, πρέπει να είναι ικανό να βασιστεί σε σωστές πληροφορίες, έτσι ώστε να εξάγει και τα σωστά συμπεράσματα. Ωστόσο, σε πολλές περιπτώσεις το σύστημα διαθέτει ελλιπείς πληροφορίες, είτε επειδή κάποια δεδομένα είναι άγνωστα για αυτό, είτε επειδή πρέπει να αντιδράσει γρήγορα και δεν έχει το χρόνο να τα συλλέξει. Επειδή οι αποφάσεις που λαμβάνονται πρέπει να είναι σωστές, είναι αναγκαίο κάποιες φορές να συμπληρώνονται και πρόσθετες πληροφορίες στη βάση γνώσης .

Μία εκδοχή για την αναπαράσταση και το σκεπτικό με βάση την κοινή λογική, αναπτύχθηκε από τον Ray Reiter και είναι γνωστή ως Default Logic (λογική με προεπιλογή). Η Default Logic είναι μία από τις πιο γνωστές προσεγγίσεις του μη μονοτονικού συλλογισμού. Η ιδέα είναι βασισμένη στη λογική πρώτης τάξης, με τη διαφορά ότι διαθέτει επιπλέον ένα σύνολο από κανόνες, ικανό να χρησιμοποιηθεί αν το συμπέρασμα δεν μπορεί να εξαχθεί άμεσα. Πλέον, με τη Default Logic, όταν το πρόγραμμα έχει ελλιπές πληροφορίες στο πρόβλημα που μελετά, θα μπορεί να κάνει εύλογες εικασίες για να φτάσει στην αλήθεια.

Η Default Logic μας απαντάει στα ερωτήματα που έχει να αντιμετωπίσει ένας πράκτορας :

1. Ποιες υποθέσεις εφαρμόζονται για να καλυφθούν τα κενά πληροφόρησης και πως χρησιμοποιούνται για την εξαγωγή συμπερασμάτων.
2. Ποια θα είναι η σχέση μεταξύ των υποθέσεων και των τωρινών (ή και μελλοντικών) συμπερασμάτων.
3. Πως θα επιβεβαιωθούν ή θα ανακατασκευαστούν οι υποθέσεις και τα συμπεράσματα, όταν μια καινούρια πληροφορία εμφανιστεί στο σύστημα.

3.2 Λειτουργία της Default Logic

Παρακάτω παραθέτουμε τη λειτουργία της Default Logic καθώς και παραδείγματα για να γίνει αυτή αντιληπτή από τον αναγνώστη, σύμφωνα με τη λογική που αυτά αποτυπώνονται στις δημοσιεύσεις των ***Aspassia Daskalopoulou και Georgios K. Giannikis*** [1,2].

Η Default Logic ακολουθεί έναν κανόνα, ο οποίος αποτελείται από ένα σύνολο προϋποθέσεων P (prerequisite), ένα σύνολο υποθέσεων J (justifications) και ένα σύνολο συμπερασμάτων C (consequent) και είναι της μορφής $P:J_1,J_2,...,J_n/C$ [1]. Τα αποτελέσματα μπορούν να συναχθούν, αν υπάρχουν προϋποθέσεις και εφόσον οι υποθέσεις ισχύουν την τρέχουσα χρονική στιγμή.

Η θεωρία της Default αποτελείται από ένα ζευγάρι (W,D), όπου W αντιστοιχεί σε ένα σύνολο από προτάσεις ή κατηγορήματα της λογικής φόρμας και το D είναι το σύνολο των προεπιλογών (defaults). Η προεπιλογή ισχύει για το κλειστό σύνολο $E \subseteq W$, αν $P \in E$ και τα $\neg J_1,\neg J_2...\neg J_n \notin E$ με τους εξής περιορισμούς:

1. Ε πρέπει να περιλαμβάνει το W
2. Το σύνολο Ε πρέπει να είναι κλειστό
3. Για ένα κανόνα που ισχύουν τα $P \in E$ και $\neg J_1,\neg J_2...\neg J_n \notin E$ τότε και $C \in E$

Έστω Π αντιπροσωπεύει μια συλλογιστική διαδικασία και καταγράφει τη σειρά με την οποία οι προεπιλογές από το D ισχύουν. Σ ε κάθε βήμα I της διαδικασίας, ο υπολογισμός περιλαμβάνει ένα σύνολο από προτάσεις $In(i) = In(i-1) \cup \{C\}$ και ένα σύνολο από υποθέσεις (που δεν πρέπει να αποδειχθούν αληθείς) $Out(i) = Out(i-1) \cup \{\neg J_1,\neg J_2...\neg J_n\}$. Έτσι το αποτέλεσμα του $\Pi(i) \cup \{Di\}$, όπου Di είναι ο κανόνας προεπιλογής του κάθε επιπέδου i [2].

Για i=0 ισχύουν : $In(0) = W$

$Out(0) = \varnothing$

$\Pi(0) = \varnothing$

Το Π(i) επιτυγχάνει όταν: $In(i) \cap Out(i) = \varnothing$, διαφορετικά αποτυγχάνει και θεωρείτε κλειστό όταν: κάθε κανόνας που ανήκει στο D και εφαρμόζεται στο In(i) ήδη συμβαίνει στο Π(i). Επί παραδείγματι :

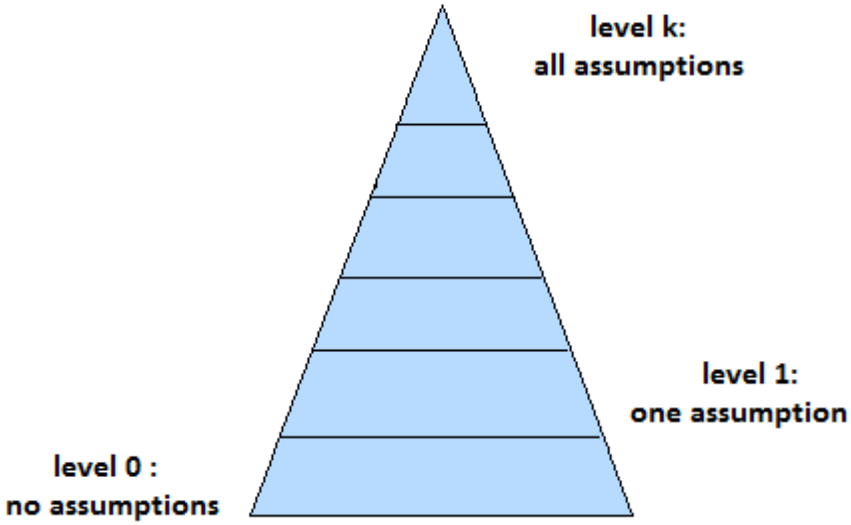
Δεδομένα	***Π(2)={D1,D2}***	***Π1={D2}***
W={A} D1≡ A : B / C D2≡ true : D / ¬B	In(2) ={A,C,¬B} Out(2) = {¬B,¬D}	In(1)={A,¬B} Out(1)={¬D}
Αποτελέσματα το Π(2) είναι κλειστό σύστημα αλλά δεν επιτυγχάνει γιατί $In(2) \cap Out(2) = \neg B \neq \varnothing$ το Π(1) είναι κλειστό σύστημα και επιτυγχάνει γιατί $In(1) \cap Out(1) = \varnothing$		

Ο πράκτορας πρέπει να εξάγει συμπέρασμα, είτε επειδή γνωρίζει όλες τις προϋποθέσεις και δεν έχει καμία υπόθεση να κάνει (assumption-free default), είτε γνωρίζοντας κάποιες προϋποθέσεις όπου και κάνει τις υποθέσεις που χρειάζεται, είτε τέλος κάνοντας μόνο υποθέσεις χωρίς να έχει στη βάση γνώσης του κάποια προϋπόθεση.

Έχοντας έναν κανόνα της μορφής : $Y \leftarrow X_1, X_2, ..., X_K$ όπου Y αντιστοιχεί στο αποτέλεσμα και το σύνολο των X αντιστοιχεί σε προϋποθέσεις ή και υποθέσεις, οι πιθανές προεπιλογές είναι 2^k.

Οι πιθανές καταστάσεις που δημιουργούνται είναι δομημένες σε μία ιεραρχία-Knowledge/Hypothesis (KH) stucture (single-KH), σε σχήμα πυραμίδας (εικόνα 2).

Κάθε επίπεδο έχει τις πιθανές προεπιλογές και περιέχει όσες πιθανές υποθέσεις χωράει. Στο επίπεδο 0 δεν υπάρχουν υποθέσεις. Όσο αυξάνονται τα επίπεδα, αυξάνονται και οι υποθέσεις, μέχρι που στο τελευταίο επίπεδο της πυραμίδας βρίσκονται και οι k υποθέσεις και δεν υπάρχει καμία προϋπόθεση για την εξαγωγή του συμπεράσματος.



εικόνα 2

Ορίζοντας ως $|L|$ το σύνολο των προεπιλογών του κάθε επιπέδου με $0 \leq L \leq k$, εξάγονται τα συμπεράσματα:

$\|L\| = 1$, όταν το επίπεδο L=0
$\|L\| = (k - L + 1) * \|L-1\| / L$, όταν το επίπεδο $L \neq 0$

Ακολουθεί ένα τυπικό παράδειγμα για k=4 καταστάσεις με 2^k=16 συνδυασμούς:

$$\mathbf{Y \leftarrow X_1, X_2, X_3, X_4}$$

ΕΠΙΠΕΔΑ	***ΠΡΟΕΠΙΛΟΓΕΣ***	***ΑΙΤΙΟΛΟΓΗΣΗ***
Level 0	$\{X_1,X_2,X_3,X_4 : \text{true}/ Y\}$	$\lvert L \rvert = 1$, καμία υπόθεση
Level 1	$\{ X_1,X_2,X_3 : X_4 /Y,$ $X_1,X_2,X_4 : X_3 /Y,$ $X_1,X_3,X_4 : X_2 /Y,$ $X_2,X_3,X_4 : X_1 /Y \}$	$\lvert L \rvert = (4 - 1 + 1) * \lvert 1 - 1 \rvert /1 = 4 * 1/1 = 4,$ μία υπόθεση
Level 2	$\{ X_1,X_2: X_4,X_3 / Y,$ $X_1,X_3: X_4, X_2 / Y,$ $X_2 ,X_3: X_4, X_1/ Y,$ $X_1,X_4: X_3 , X_2 /Y,$ $X_2 ,X_4: X_3, X_1/ Y,$ $X_3,X_4 : X_2,X_1 / Y \}$	$\lvert L \rvert = (4 - 2 + 1) * \lvert 2 - 1 \rvert /2 = 3 * 4/2 = 6,$ δύο υποθέσεις
Level 3	$\{ X_1: X_4,X_3,X_2 / Y,$ $X_2: X_4,X_3,X_1/ Y,$ $X_3: X_4,X_2,X_1 / Y,$ $X_4: X_3,X_2,X_1 / Y \}$	$\lvert L \rvert = (4 - 3 + 1) * \lvert 3 - 1 \rvert /3 = 2 * 6/3 = 4,$ τρεις υποθέσεις
Level 4	$\{\text{true}: X_4,X_3,X_2,X_1 / Y\}$	$\lvert L \rvert = (4 - 4 + 1) * \lvert 4 - 1 \rvert /4 = 1 * 4/4 = 1,$ τέσσερις υποθέσεις

Δεδομένου ότι η βάση γνώσης μπορεί να περιέχει παραπάνω από έναν κανόνα, για κάθε έναν από αυτούς δημιουργείται η παραπάνω δομή single-KH. Όλα τα αποτελέσματα των κανόνων συντίθενται σε ένα πολύγωνο (εικόνα 3), το οποίο περιέχει τόσα επίπεδα, όσα το ψηλότερο από τις δομές των single-KH και αποκαλείται multi-KH structure.

Δίνοντας ένα σύνολο από κανόνες, ο αριθμός των επιπέδων του multi-KH θα είναι ίσος με το μέγιστο k_i και κάθε επίπεδο θα έχει το σύνολο των προεπιλογών των αντίστοιχων επιπέδων single-KH. Για την ακρίβεια ένα multi-KH δεν έχει ένα ενιαίο τελευταίο επίπεδο, δεδομένου ότι κάθε κανόνας έχει τη δική του κορυφή. Μας ενδιαφέρει το υψηλότερο από όλα επίπεδο, γιατί σε εκείνο το σημείο τερματίζει η διαδικασία. [(3)]

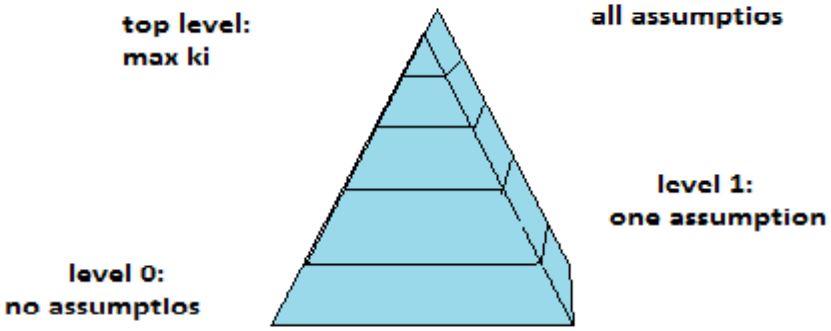
[(3)]Στην περίπτωση που χρησιμοποιηθούν κανόνες μόνο από το επίπεδο 0, δηλαδή να μην γίνει χρήση υποθέσεων, τότε η λύση είναι ταυτόσημη με αυτήν της κλασικής λογικής.

Ακολουθεί ένα παράδειγμα όπου οι κανόνες είναι περισσότεροι από έναν, έστω:

R1≡ Y1←X_1,X_2

R2≡ Y1←X_3,X_4, X5

ΕΠΙΠΕΔΑ	*Single-norm KH structure (R1)*	*Single-norm KH structure (R2)*	*Multi-norm KH structure (R1,R2)*
Level 0	{$D1_{0,1}$≡X1,X2:true/Y1 }	{$D2_{0,1}$≡X3,X4,X5 : true/Y2 }	{$D1_{0,1}$≡X1,X2:true/Y1 , $D2_{0,1}$≡X3,X4,X5 : true/Y2 }
Level 1	{$D1_{1,1}$≡X1:X2/Y1, $D1_{1,2}$≡X2:X1/Y1 }	{$D2_{1,1}$≡X3,X4:X5/Y2, $D2_{1,2}$≡X3,X5:X4/Y2 , $D2_{1,3}$≡X4,X5:X3/Y2}	{$D1_{1,1}$≡X1:X2/Y1, $D2_{1,1}$≡X3,X4:X5/Y2, $D1_{1,2}$≡X2:X1/Y1, $D2_{1,2}$≡X3,X5:X4/Y2 , $D2_{1,3}$≡X4,X5:X3/Y2 }
Level 2	{$D1_{2,1}$≡true :X1,X2/Y1 }	{$D2_{2,1}$≡X3:X4,X5/Y2, $D2_{2,2}$≡X4:X5,X3/Y2 , $D2_{2,3}$≡X5:X4,X3/Y2}	{$D1_{2,1}$≡true :X1,X2/Y1, $D2_{2,1}$≡X3:X4,X5/Y2, $D2_{2,2}$≡X4:X5,X3/Y2 , $D2_{2,3}$≡X5:X4,X3/Y2}
Level 3		{$D2_{3,1}$≡true:X5,X4,X3/Y2}	{$D2_{3,1}$≡true:X5,X4,X3/Y2}



εικόνα 3

Παρατηρείται ότι το υψηλότερο (και τελευταίο) επίπεδο του multi-KH structure είναι το τρίτο, όσο δηλαδή και του R2 και περιέχει μόνο την προεπιλογή αυτού καθώς το R1 στο επίπεδο 3 δεν έχει κανένα default.

3.2.1 Προτεραιότητες

Στην παρούσα φάση πρέπει να οριστούν οι προτεραιότητες τόσο του single-KH όσο και του multi-KH. Οι ***Aspassia Daskalopoulou και Georgios K. Giannikis*** [1,2] μας δίνουν σαφή εικόνα για τον τρόπο με τον οποίο λαμβάνονται οι προτεραιότητες.

Με βάση το σχήμα της πυραμίδας και του πολυγώνου, μεγαλύτερη προτεραιότητα έχουν τα χαμηλότερα επίπεδα έναντι των υψηλότερων, καθώς διαθέτουν λιγότερες υποθέσεις. Έτσι το επίπεδο 0 έχει υψηλότερη προτεραιότητα έναντι του πρώτου, αυτό έναντι του δεύτερου και ούτω καθεξής.

Όταν δύο ή και περισσότερες προεπιλογές βρίσκονται στο ίδιο επίπεδο, της ίδιας δομής single-KH (για παράδειγμα τα $D2_{2,1}$,$D2_{2,2}$, $D2_{2,3}$) εφόσον έχουν τον ίδιο αριθμό υποθέσεων θα έχουν και την ίδια προτεραιότητα.

Τέλος όταν δύο ή και περισσότερες προεπιλογές βρίσκονται στο ίδιο επίπεδο της δομής multi-KH (για παράδειγμα όπου $W=\{X_1,X_3,X_4\}$, κάνοντας την υπόθεση ότι X_2 και X_5 ισχύουν, παρατηρούμε ότι το $In(2) = W \cup \{Y_1,Y_2\}$ και $Out(2) = \{\neg X_2, \neg X_5\}$ με προεπιλογές τις $D1_{1,1}$ και $D2_{1,1}$. Και οι δύο προεπιλογές έχουν τον ίδιο αριθμό υποθέσεων) υψηλότερη προτεραιότητα εμφανίζει το $D2_{1,1}$ γιατί προέρχεται από μεγαλύτερη βάση από ότι το $D1_{1,1}$.

4

Σχεδιασμός και Υλοποίηση

4.1 Σχεδίαση

Στο κεφάλαιο αυτό αναφέρουμε τον τρόπο δημιουργίας της βιβλιοθήκης μας μέσω της οποίας προσπαθούμε να προσφέρουμε την δυνατότητα υποστήριξης υποθέσεων σε γραμματική τύπου Default Logic. Ο σχεδιασμός μας βασίστηκε στο θεωρητικό υπόβαθρο όπως αυτό αναλύθηκε στο *Κεφάλαιο 3*.

Ξεκινώντας από μία υπάρχουσα γραμματική και κάνοντας χρήση του κανόνα *rule(Goal, P, J, C)* ο χρήστης μπορεί να θέσει ερωτήσεις οι οποίες απαντώνται με ή χωρίς υποθέσεις. Οι πιθανές απαντήσεις είναι:

- “definitely true”
- “definitely false”
- “presumably true”
- “presumably false”

Για τις δύο τελευταίες απαντήσεις η έξοδος θα συμπεριλαμβάνει και τις υποθέσεις που έγιναν κατά την διερεύνηση της ερώτησης εν αντιθέσει με τις δύο πρώτες όπου δεν θα χρησιμοποιούν υποθέσεις για την εξαγωγή συμπεράσματος. Σε όλες τις απαντήσεις αναφέρονται οι κανόνες που χρησιμοποιήθηκαν ως προϋποθέσεις για την εξαγωγή του συμπεράσματος. Πιο συγκεκριμένα στη λίστα P (Prerequisites) βρίσκονται οι προϋποθέσεις, στην J (Justifications) οι υποθέσεις και στην C (Conclusions) το συμπέρασμα της ερώτησης.

Ο κανόνας *rule(Goal, P, J, C)* γίνεται διαθέσιμος ενσωματώνοντας τη βιβλιοθήκη μας κάνοντας χρήση του *consult('assumptions.pl')* (παράρτημα Β) στο αρχείο που περιλαμβάνει την αρχική γραμματική.

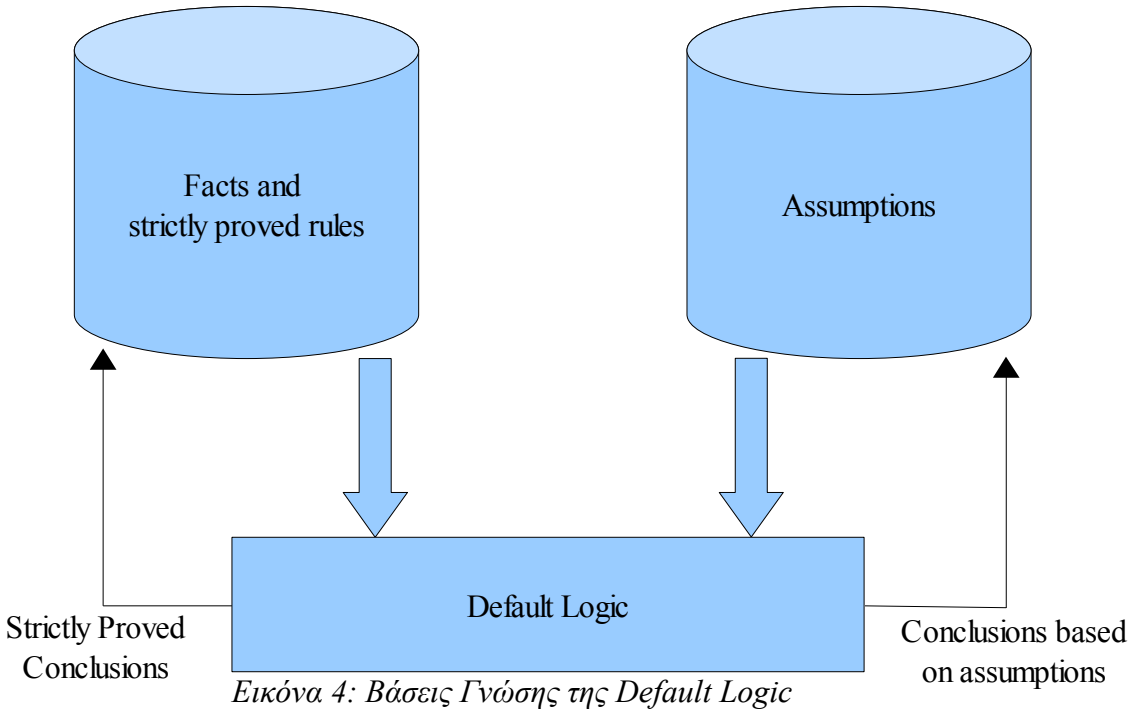
4.1.1 Λήψη Υποθέσεων

Έχουμε εντοπίσει πως ως υποθέσεις μπορούν να χαρακτηριστούν κανόνες υπό τρεις συνθήκες:

1. Η ερώτηση αφορά κανόνα, για τον οποίο δεν έχουμε γνώση, δηλαδή δεν υπάρχει στην αρχική γραμματική.
2. Η ερώτηση αφορά κανόνα, ο οποίος στο σώμα του βασίζεται σε απροσδιόριστες μεταβλητές. Δηλαδή μεταβλητές οι οποίες δεν έχουν λάβει τιμή, κάτι το οποίο είναι πιθανό όταν στην ροή του προγράμματος υπεισέρχονται υποθέσεις.
3. Η ερώτηση αφορά κανόνα, ο οποίος στο σώμα του βασίζεται σε υποθέσεις. Έτσι είναι ασφαλές να θεωρηθεί ο εξεταζόμενος κανόνας ως υπόθεση.

Σε αντίθετη περίπτωση η λίστα υποθέσεων για μία δεδομένη ερώτηση είναι κενή και το συμπέρασμα είναι βέβαιο (“definitely true”, “definitely false”).

Στο ακόλουθο σχήμα αποτυπώνεται η λήψη συμπερασμάτων σε γνωσιακές βάσεις:



Εικόνα 4: Βάσεις Γνώσης της Default Logic

4.2 Εγχειρίδιο χρήσης

Κατά τον σχεδιασμό του εργαλείου μας ως στόχο είχαμε να μην επιβαρύνουμε τον τελικό χρήστη αναγκάζοντάς τον να κάνει σημαντικές αλλαγές στη γραμματική που θέλει να εφαρμόσει υποθέσεις. Ο χρήστης καλείται να ενσωματώσει την λειτουργία των υποθέσεων γράφοντας στην αρχή της γραμματικής του το *consult('assumptions.pl').* Αυτό είναι αρκετό για να χρησιμοποιήσει στην συνέχεια τον κανόνα *rule(Goal, P, J, C)* ο οποίος αναφέρει τις προϋποθέσεις, υποθέσεις και συμπεράσματα της ερώτησης *Goal* σύμφωνα με τη λογική που περιγράψαμε στη *παράγραφο 5.1.1*.

Κατά την ανάπτυξη της βιβλιοθήκης μας παρατηρήσαμε ότι οι υποθέσεις σε συγκεκριμένα σενάρια οδηγούν σε μη λογικά αποτελέσματα. Έτσι δίνουμε την επιλογή στον χρήστη να αφαιρέσει την λειτουργία υποθέσεων σε μεμονωμένους κανόνες μέσω της επιλογής *do_not_assume(predicate)*.

Για περαιτέρω πληροφορίες σχετικά με τον τρόπο χρήσης της πλατφόρμας SWI-Prolog ανατρέξτε στο *παράρτημα Α*.

4.3 Λεπτομέρειες Υλοποίησης

Ο κανόνας *rule(Goal, P, J, C)* εσωτερικά βασίζεται στον κανόνα *assume(Goal, J, Success)*. Για κάθε ανάκληση του πρώτου ο δεύτερος εφαρμόζεται στο σώμα του (αν αυτό υπάρχει) και εκτελείται αναδρομικά για την παραγωγή υποθέσεων όπου αυτές προκύπτουν βάση των κανόνων που αναφέραμε στην *παράγραφο 5.1.1*.

Για κάθε ερώτηση διατηρούνται 2 βάσεις πληροφορίας. Η μία είναι αυτή που υπάρχει εκ σχεδιασμού στην Default Logic και περιέχει τους κανόνες οι οποίοι αποδεικνύονται κάνοντας χρήση γεγονότων και αυστηρά αποδεικνυόμενων κανόνων. Αυτή συμπεριλαμβάνει δηλαδή τις αρχικές δηλώσεις γνώσεως που υπάρχουν στη γραμματική του χρήστη καθώς και όλους τους κανόνες οι οποίοι προκύπτουν από αυτές. Παράλληλα διατηρείται μία βάση γνώσεως σε μορφή λίστας η οποία περιέχει τις υποθέσεις που παράχθηκαν κατά τη διερεύνηση ενός κανόνα. Εδώ είναι σημαντικό να αναφέρουμε πως ένα *assume(Goal, J, Success)* μπορεί να συμπεράνει ότι το *Goal* δεν μπορεί να αποφανθεί ως επιτυχές. Η επιτυχία ή όχι υποδεικνύεται από την τιμή της μεταβλητής *Success*. Αν είναι *true* τότε αποδείχθηκε πως το *Goal* ισχύει, διαφορετικά για *Success=false* το *Goal* δεν ισχύει. Και στις δύο περιπτώσεις η λίστα J εμπεριέχει τις υποθέσεις που τυχόν παρήχθησαν. Επιπρόσθετα αν η απόφαση για το *Goal* βασίστηκε σε υπόθεση συμπεριλαμβάνεται και αυτό στη λίστα J. Στην ίδια λογική ο κανόνας *rule* προσθέτει το *Goal* ή το *not(Goal)* στη λίστα των συμπερασμάτων ανάλογα με το αν ισχύει ή όχι η ερώτηση.

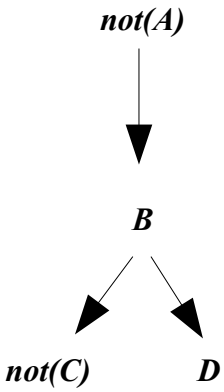
Μία σημαντική λεπτομέρεια είναι πως οι κανόνες που εμπεριέχονται σε *not* μεταχειρίζονται σύμφωνα με τον παρακάτω ψευδοκώδικα:

```
assume(Goal, J, Success):-
        Goal=not(NegativeGoal),
        assume(NegativeGoal, Assumptions, NegativeSuccess),
        J = Assumptions,
        Success=not(NegativeSuccess).
```

Αυτό γίνεται για να υποστηριχθούν εμφωλευμένα *not* για τα οποία διατηρούνται όλες οι υποθέσεις όπως αυτό που εμφανίζεται στο παρακάτω αφηρημένο δένδρο:

Παράδειγμα εμφωλευμένων not

Αφηρημένο δένδρο



Στο σχήμα βλέπουμε το σκεπτικό μας για την διατήρηση των εμφωλευμένων *not* στις υποθέσεις μας. Έτσι ξεκινώντας από το not(C), ελέγχουμε αν ισχύει το C. Αν ισχύει το C μπαίνει στη λίστα των υποθέσεων μας και το not(C) πλέον είναι false. Ανεβαίνοντας στο B (είτε ισχύει είτε όχι το D) το B θα είναι false. Έτσι οδηγούμαστε στο A όπου είναι και αυτό false(λόγω του B) και το not(A) εξάγεται ως true. Να σημειωθεί ότι το [C] υφίσταται ως υπόθεση σε όλα τα επίπεδα (από εκεί που ορίστηκε έως και στο επίπεδο εξαγωγής συμπεράσματος.

4.4 Παραδείγματα Τρεξίματος

Ο χρήστης πρέπει να δώσει μία αρχική γραμματική στην οποία θα έχει ενσωματώσει τη λειτουργία των υποθέσεων κάνοντας *:-consult('assumptions.pl')*. Για εμάς η αρχική αυτή γραμματική είναι ένα *σενάριο* επικοινωνίας μεταξύ ενός buyer και ενός seller το οποίο αναλύεται στο ***Παράρτημα Γ***.

1) Παράδειγμα στο οποίο έχουμε γνώση για όλα τα δεδομένα

?- rule(pay(maria,nero,1,kurNikos,E), P,J,C).

definitely true

E = 1,

P = [send_price(kurNikos, nero, 1, maria, 1), money(maria, _G2626), _G2626>=1, !],

J = [],

C = [pay(maria, nero, 1, kurNikos, 1)].

Παρατηρούμε ότι η απάντηση είναι "definitely true" ("σίγουρα ναι") σε έναν κανόνα όπου είναι γνωστά όλα τα δεδομένα. Για τον λόγο αυτό άλλωστε η λίστα των υποθέσεων είναι κενή (J=[]), η λίστα των προϋποθέσεων περιέχει το *"Body"* του κανόνα pay και στη λίστα των συμπερασμάτων (C) μπαίνει πλέον η ερώτηση που θέσαμε, αφού αποδείχθηκε.

2) Παράδειγμα στο οποίο τα δεδομένα δεν είναι γνωστά

5 ?- rule(pay(ann,tsai,1,mike,E), P,J,C).

presumably true

E = 1,

P = [],

J = [pay(ann, tsai, 1, mike, 1), (_G1072+_G1075>=1, !), money(ann, _G1072+_G1075), salary(ann, _G1075), bank(ann, _G1072), send_price(mike, tsai, 1, ann, 1), 1 is 1*_G72, cost(tsai, _G72), order(..., ..., ..., ...)|...],

C = [pay(ann, tsai, 1, mike, 1)].

Σε αυτό το παράδειγμα τα δεδομένα μας (ann, tsai , mike) ,δεν είναι γνωστά. Η λίστα των προϋποθέσεων (P) είναι κενή και όλοι οι κανόνες από το "Body" του pay γίνονται υποθέσεις και μπαίνουν μέσα στη λίστα J , έτσι ώστε να πάρουμε πιθανολογική απάντηση και η ερώτηση να απαντηθεί με "presumably true" ("πιθανότατα ναι").

3α) Παράδειγμα με γνωστά μερικά δεδομένα-presumably

?- rule(consider_buying(cathy,nero,1), P,J,C).

presumably true

P = [missing(cathy, nero, 1)],

J = [consider_buying(cathy, nero, 1), have_money_for(cathy, nero, 1), _G2364+_G2367>=1*1, money(cathy, _G2364+_G2367), salary(cathy, _G2367), bank(cathy, _G2364)],

C = [consider_buying(cathy, nero, 1)].

Εδώ παρατηρούμε ότι στις προϋποθέσεις υπάρχει το missing(cathy,nero,1) , αφού στο "Body" του missing έχουμε ορίσει ότι **όλοι** έχουν ανάγκη από **"κάτι"**,(παρόλο που το όνομα Cathy δεν είναι αρχικοποιημένο-γνωστό). Στη λίστα των υποθέσεων μπαίνουν τα : consider_buying, have_money_for, money, salary, bank) και θεωρούμε ότι ισχύουν . Η απάντηση στην ερώτηση αυτή είναι "persumably true".

3β) Παράδειγμα με βεβαιότητα αρνητικό

?- rule(consider_buying(liza,nero,1),P,J,C).

definitely false

P = [missing(liza, nero, 1), have_money_for(liza, nero, 1)],

J = [],

C = [not(consider_buying(liza, nero, 1))].

Στο παρόν παράδειγμα η liza δεν μπορεί να σχεδιάσει να πάρει νερό, καθώς δεν έχει λεφτά (από τη database bank(liza,0) και salary(liza,0)). Η λίστα των προϋποθέσεων περιέχει τα : missing(liza,nero,1) , have_money_for(liza,nero,1) , όμως η liza δεν έχει λεφτά και αυτός είναι ο λόγος που δεν γίνεται υπόθεση και η απάντηση μας είναι " definitely false " (" σίγουρα λάθος ").

4) Παράδειγμα με βεβαιότητα αρνητικό-χρήση do_not_assume()

?- rule(consider_buying(nikos,nero,1), P,J,C).

definitely false

P = [missing(nikos, nero, 1), have_money_for(nikos, nero, 1)],

J = [],

C = [not(consider_buying(nikos, nero, 1))].

Στο παραπάνω παράδειγμα θέτουμε την ερώτηση αν ο Νίκος θα σκεφτεί να αγοράσει νερό. Στην αρχική database έχει οριστεί ότι ο Νίκος έχει τουλάχιστον ένα νερό : ***have_at_least(nikos, nero, 1).*** Όμως όπως γίνεται αντιληπτό από το πρόγραμμά μας με τη χρήση του : ***do_not_assume(have_at_least) ,*** δεν επιτρέπουμε να γίνουν υποθέσεις σε κάποιον που έχει ***"have at least".*** Έτσι ο Νίκος δεν θα σκεφτεί να αγοράσει και για το λόγο αυτό, η λίστα των υποθέσεων είναι κενή (J[]) και το συμπέρασμα που εξάγεται είναι *not(consider_buying(nikos, nero, 1)).* Η τελική μας απάντηση είναι το "*definitely false*" ("σίγουρα λάθος").

5

Επίλογος

5.1 Σύνοψη και συμπεράσματα

Η εργασία που παρουσιάζουμε υπαγόρευε την ανάγκη δημιουργίας υποθέσεων στους πράκτορες λογισμικού, κάνοντας χρήση κανόνες της Default Logic. Η συμπεριφορά των πρακτόρων σε περιβάλλοντα όπου αλληλεπιδρούν με άλλους πράκτορες, περιορίζεται από κανόνες που ρυθμίζουν το περιβάλλον στο οποίο συμμετέχουν.

Στην παρούσα εργασία ο στόχος μας επετεύχθη. Ένας πράκτορας, θέλοντας να σχεδιάσει τις κινήσεις του, είναι ικανός να απαντήσει σε οποιαδήποτε ερώτηση είτε ρητά (καταφατικά/αρνητικά) είτε κάνοντας χρήση υποθέσεων.

Η αυτονομία κάθε πράκτορα έγκειται στο γεγονός του κατά πόσο είναι " ελεύθερος " να κάνει υποθέσεις για πράγματα που δεν γνωρίζει. Σε αυτό το σημείο εμφανίζεται και η πρώτη μας αδυναμία. Ο πράκτορας μας όταν έχει άγνοια για κάποια ενέργεια υποθέτει ότι ισχύει, έτσι ώστε να καταλήξει σε ένα συμπέρασμα (*σωστό/λάθος/πιθανά σωστό/πιθανά λάθος*). Η απόφαση να θεωρηθεί ότι μία ενέργεια μάλλον είναι σωστή ώστε να συνεχιστεί ο συλλογισμός, δεν καθίσταται και αξιόπιστη. Δεν είναι ικανός να αξιολογήσει ποιες αποφάσεις κινούνται στα πλαίσια της λογικής και ποιες όχι και αυτό μπορεί να δημιουργήσει προβλήματα σε σημαντικές εφαρμογές όπως αν χρησιμοποιηθεί για αναζήτηση στο διαδίκτυο, έλεγχο εργοστασιακών μονάδων και παροχή έξυπνων υπηρεσιών βοήθειας,

Μία άλλη αδυναμία του προγράμματός μας είναι ότι δεν είναι σε θέση να υπολογίσει λογικές πράξεις. Έτσι για μία ερώτηση (G>1,G<1) η απάντηση που θα πάρουμε είναι " πιθανόν σωστή " , αποτέλεσμα το οποίο δεν είναι επιθυμητό.

5.2 Μελλοντικές Επεκτάσεις

Όπως αναφέρθηκε στο *κεφάλαιο 5.1* ως μελλοντική εργασία θα μπορούσε να καλυφθεί η αδυναμία υπολογισμού λογικών πράξεων. Έτσι φανταζόμαστε την επέκταση του προγράμματος μας, ώστε να υποστηρίζει κανόνες οι οποίοι θα είναι σε θέση να ελέγχουν και να υπολογίζουν τις λογικές αυτές πράξεις.

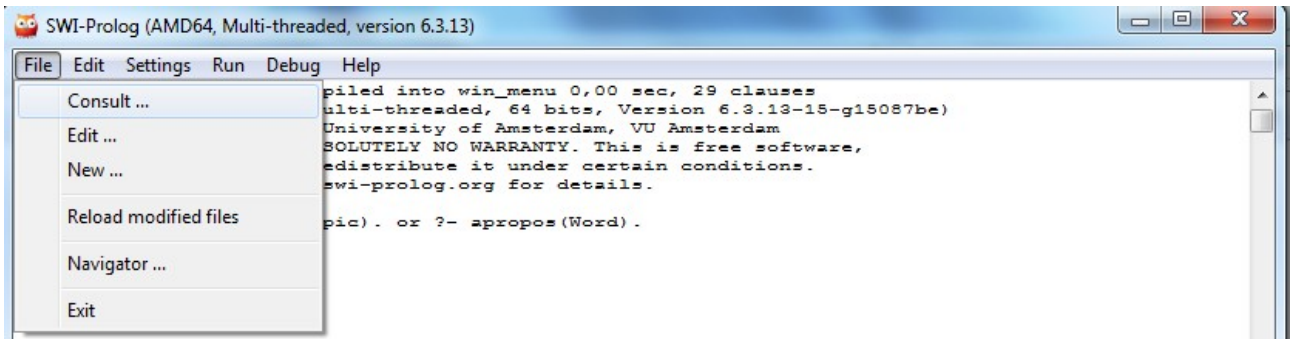
Άλλη μία επέκταση του προγράμματος θα μπορούσε να είναι ο έλεγχος της αξιοπιστίας των υποθέσεων που θα λάβει ο πράκτορας. Αν μπορέσει να αξιολογήσει ποιες υποθέσεις είναι λογικές και ποιες όχι, οι ενέργειες του θα είναι πιο σωστές και το πρόγραμμα κατά πολύ πιο αξιόπιστο.

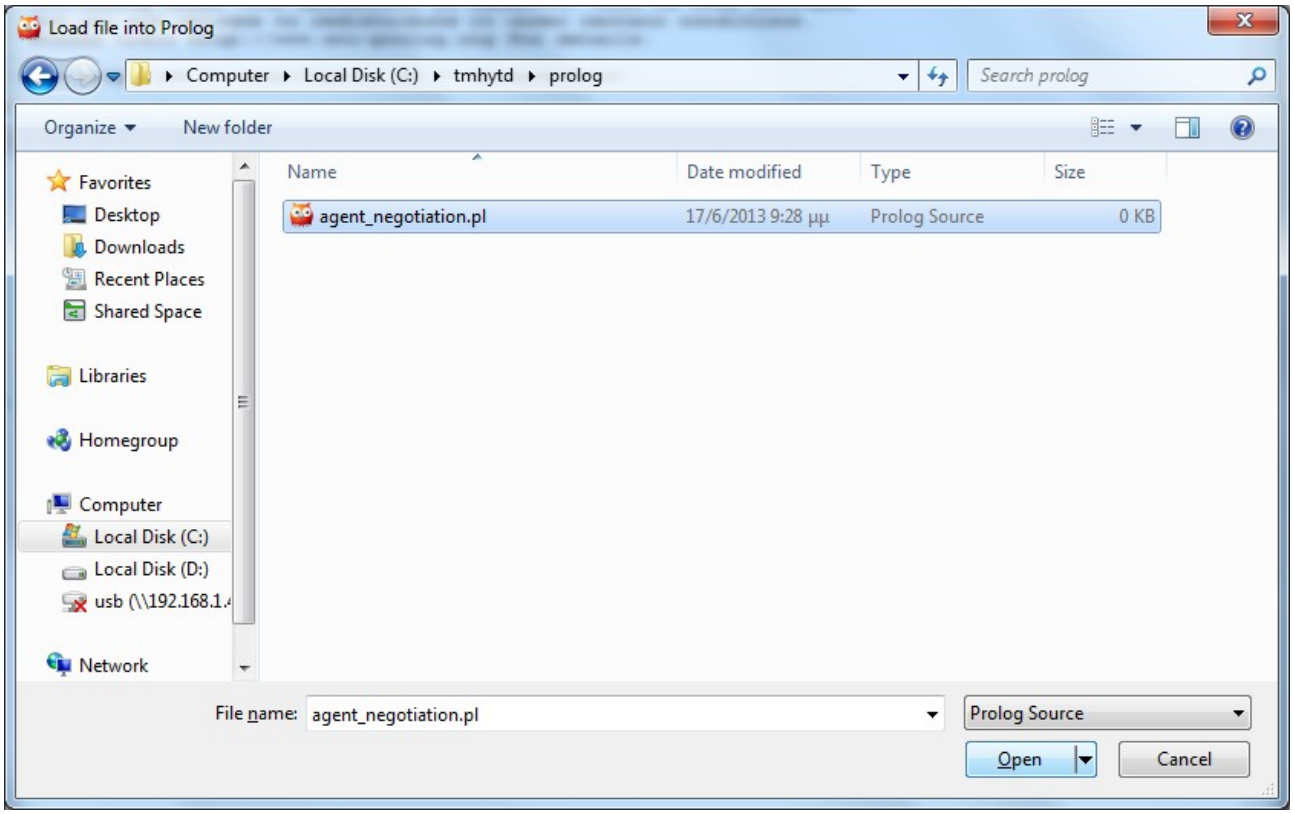
Τέλος ως μελλοντική δουλειά θα μπορούσε να είναι η εξέταση του κατά πόσο η προσέγγισή μας θα μπορούσε να λειτουργήσει ικανοποιητικά σε πολυπρακτορικά συστήματα, στα οποία γίνονται πολλές και ταυτόχρονες συναλλαγές καθώς και στο κατά πόσο θα μπορούσε να εφαρμοστεί η συλλογιστική μας σε άλλες προσεγγίσεις μη-μονοτονικού συλλογισμού όπως Logic Programs[7,8] και Defeasible Logic [9].

Παράρτημα Α

Η πλατφόρμα SWI-Prolog μπορεί να εγκατασταθεί αφού κατεβάσουμε το πακέτο εγκατάστασης από τη σελίδα http://www.swi-prolog.org/.

Η πλατφόρμα αποτελείται από ένα περιβάλλον εκτέλεσης το οποίο συμπληρώνεται από εργαλεία ανάπτυξης των εφαρμογών αλλά και αποσφαλμάτωσης. Η κατάληξη των αρχείων Prolog που διαχειρίζονται από την εφαρμογή είναι ".pl". Για την εκτέλεση αυτών αφού ανοίξουμε την εφαρμογή swi-prolog επιλέγουμε από το menu File την εντολή Consult και στο παράθυρο που ανοίγει εντοπίζουμε και ανοίγουμε το αρχείο .pl της γραμματικής.





Στη συνέχεια για την υποβολή ερωτήσεων γράφουμε το query ακολουθούμενο από "." και πατάμε το πλήκτρο *Enter.*

Η λειτουργία αποσφαλμάτωσης ενεργοποιείται γράφοντας στο τερματικό της SWI-Prolog *gdebug, gtrace*. Εν συνεχεία για όλες τις επόμενες ερωτήσεις που θα τεθούν θα ενεργοποιηθεί ο ενσωματωμένος debugger και οι ενδιάμεσες καταστάσεις θα αναλύονται βήμα-βήμα στο καινούριο παράθυρο που θα αναδυθεί. Για να προχωρήσει η εκτέλεση στον επόμενο κανόνα κατά τη διαδικασία του debugging χρειάζεται να πατήσουμε το πλήκτρο *space*.

Παράρτημα Β

Η βιβλιοθήκη μας assumptions.pl

```
:- multifile (rule)/4, assume/3.
:- dynamic do_not_assume/1.

kati2(1):- true.
test4(A,B) :- not(test3(A,B)).
test3(A,B) :- not(test(A,B)), not(kati2(2)), not(A==B).
test2(A,B) :- not(test(A,B)), test(A,B), A==B.
%test(A, B) :- A == B.

tree_to_list( (A, B), L) :- tree_to_list(B, Rest), append([A], Rest, L).
tree_to_list( A, L) :- not(functor(A, ',', _)), append([], [A], L).

and(true, true, true):- !.
and(_, _, false) :- !.

filter([H|T], List2, List) :- member(H, T), filter(T, List2, List), !.
filter([H|T], List2, List) :- member(H, List2), filter(T, List2, List), !.
filter([H|T], List2, List) :- not(member(H, T)), not(member(H, List2)), filter(T, List2, List1),
append([H], List1, List), !.
filter([], _, []) :- !.
filter(List, [], List):- !.

%rule(Goal, P, J, C) :-  not(predicate_property(Goal,undefined)), predicate_property(Goal,
number_of_rules(_)),
%                                                                    %not(Goal),
clause(Goal, Body), assume(Body, J, Success),
%                                                                    (Success-
>append([], [Goal], C) ; append([], [not(Goal)], C)),  tree_to_list(Body, P1), filter(P1, J, P),
%                                                                    (append([], [],
J) -> write('definitely ') ; write('presumably ')), write(Success),!.
rule(Goal, P, J, C) :-   %not(predicate_property(Goal,undefined)),
            assume(Goal, J,Success),
            catch(clause(Goal, Body), _, fail),
           (Success->append([], [Goal], C) ; append([], [not(Goal)], C)),
           tree_to_list(Body, P1),
```

```
        filter(P1, J, P),
        (append([], [], J) -> write('definitely ') ; write('presumably ')), write(Success),!.

rule(Goal, [], J, C) :-   %not(predicate_property(Goal,undefined)),
         assume(Goal, J,Success),
        (Success->append([], [Goal], C) ; append([], [not(Goal)], C)),
        (append([], [], J) -> write('definitely ') ; write('presumably ')), write(Success),!.

% Diatrexei ena pros ena to dendro ( andromika to upodendro ) pou sun8etei to deksi kommati enos kanona
assume((A,B), J, Success) :-
        assume(A,J2,Success1),
        assume(B, J1, Success2),
        and(Success1, Success2, Success),
        append(J1, J2, J).

% Otan o kanonas exei sta8eres ws orismata ta opoia einai diaforetika apo auta pou exw mesa sto A
% Epishs den einai orismeno to do_not_assume(predicate_tou_A)
assume(A, J, Success) :-
        not(functor(A,',', _)),
        not(functor(A,'not', _)),
        not(predicate_property(A,undefined)),
        predicate_property(A, number_of_rules(_)),
        not(A), not(clause(A, _)),
        functor(A, Functor, _),
        (do_not_assume(Functor)->(Success=false,append([], [], J));(Success=true,append([], [A], J))).

% Autos o kanonas diaxeirizetai ta not ( *kanenas* allos kanonas den mporei na ginei true otan to orisma tou einai tou typou not(Predicate) )
assume(A, J, Success) :-
        functor(A,'not', _),
        A=not(B),
        assume(B, J1, Success1),
        (append([],[], J1) -> append([],[],J) ; append(J1,[], j)),
        (Success1 -> Success=false ; Success=true).

% Gia predicates pou den einai orismena
assume(A, J, true) :-
```

```
        predicate_property(A,undefined),
        not(functor(A,'not', _)),
        not(functor(A,',', _)),
        append([], [A], J).

assume(A, J, true) :-
        var(A),
        append([], [A], J). % gia na kalupsw thn periptwsh px  tou assert(try_harder) wste to assume(try_harder) na isxuei

assume(A, J, Success) :-
        not(var(A)),
        predicate_property(A, built_in),
        A=..[_, _,_],
        catch( (A, append([],[],J), Success=true), _, fail).

% auto 8a kanei upo8esh gia to G<1
assume(A, J, Success) :-
        not(var(A)),
        predicate_property(A, built_in),
        A=..[_, _,_],
        catch( (not(A), append([],[],J), Success=false), _, (append([], [A], J), Success=true, !)).

% Mesw tou catch prospa8w na apodeiksw to A, an uparksei kapoio exception prospa8w na apodeiksw mia paragwgh tou
assume(A, J, Success) :-
        not(functor(A,'not', _)),
        not(functor(A,',', _)),
        predicate_property(A, number_of_rules(_)),
        catch( (A,append([], [], J),Success=true) , _, ( clause(A, B), assume(B, J1, Success), (not(append([],[], J1))->(Success->append([A], J1, J);append([not(A)], J1, J));append([],[],J)))).

assume(A, J, Success) :-
        not(functor(A,',', _)),
        not(functor(A,'not', _)),
        not(predicate_property(A,undefined)),
        predicate_property(A, number_of_rules(_)),
        not(A),
        clause(A, Body),
```

```
        functor(A, Functor, _),
        not(do_not_assume(Functor)),
        assume(Body, J1, Success),
        (not(append([],[], J1))->(Success->append([A], J1, J);append([not(A)], J1, J));append([],
[],J)).
```

Παράρτημα Γ

Στο σενάριο που παραθέτουμε υπάρχουν 2 ειδών πράκτορες οι αγοραστές και πωλητές. Αυτοί αλληλεπιδρούν μεταξύ τους με σκοπό την αγοραπωλησία προϊόντων. Η βάση γνώσης μας περιέχει τα εξής γεγονότα:

- Αρχική γνώση για τα προϊόντα και την ποσότητα αυτών που διαθέτει ο εκάστοτε πωλητής.
- Τιμές για τα προϊόντα.
- Ανάγκες που εμφανίζουν οι αγοραστές.
- Προϊόντα που ικανοποιούν τις παραπάνω ανάγκες.
- Ποια προϊόντα έχουν ήδη στην κατοχή τους οι αγοραστές.
- Τα χρήματα που διαθέτει ο εκάστοτε αγοραστής.

Ακολουθούν οι κανόνες που διέπουν την διάδραση μεταξύ των πρακτόρων.
Από τη σκοπιά του αγοραστή:

1. Αρχικά εξετάζει αν έχει έλλειψη κάποιου προϊόντος το οποίο ικανοποιεί κάποια ανάγκη του.
2. Για προϊόντα που εντοπίζει έλλειψη μπαίνει στη διαδικασία να σκεφτεί την αγορά τους, εάν διαθέτει το απαραίτητο ποσό.
3. Σε περίπτωση που ικανοποιούνται οι παραπάνω συνθήκες θα ρωτήσει έναν πωλητή για την διαθεσιμότητα του προϊόντος που τον ενδιαφέρει.
4. Για θετική απόκριση από έναν πωλητή, στέλνει παραγγελία και ζητά την συνολική τιμή
5. Η πληρωμή θα πραγματοποιηθεί μόλις λάβει την τιμή των προϊόντων από τον πωλητή και σιγουρευτεί πως διαθέτει το απαιτούμενο ποσό.

Από τη σκοπιά του πωλητή:

1. Μετά από ερώτηση διαθεσιμότητας προϊόντων από κάποιον αγοραστή απαντά με τον αριθμό των προϊόντων στην αποθήκη του εάν και εφόσον έχει.
2. Μετά από την αίτηση παραγγελίας ενός αγοραστή τον ενημερώνει για το συνολικό κόστος που έχει υπολογίσει.

```
%Arxikopoihseis
:- consult('assumptions.pl').

stock(kurNikos,nero,1000).
stock(kurNikos,kafe,500).
satisfy_need(_,upnhlia,kafe,1).
satisfy_need(_, dipsa, nero, 1).
have_need(_,upnhlia).
have_need(_, dipsa).
have_at_least(nikos, nero, 1).
do_not_assume(have_at_least).
bank(nikos, 1000).
bank(liza,0).
bank(makhs, 1).
bank(maria,30000).
bank(lamprini,1000000).
salary(liza,0).
salary(maria,0).
salary(makhs,1200).
salary(nikos, 0).
cost(nero, 1).
cost(kafe, 4).
can_offer(kurNikos, nero).
can_offer(kurNikos,kafe).

% Expr = Arithmos proiontwn
missing(Buyer, Product, ExprProduct) :- not( have_at_least(Buyer, Product, ExprProduct)),
        satisfy_need(Buyer, Need, Product, ExprProduct),
        have_need(Buyer, Need).

% Expr = Arithmos Proiontwn
consider_buying(Buyer, Product, ExprProduct) :- missing(Buyer, Product,
ExprProduct),have_money_for(Buyer, Product, ExprProduct).

% Expr = Arithmos proiontwn
have_money_for(Buyer, Product, ExprProduct) :- cost(Product, ExprCost),
        money(Buyer, ExprMoney),
        ExprMoney >= ExprCost*ExprProduct.
```

```
money(Buyer, Expr) :- bank(Buyer, Expr1), salary(Buyer, Expr2), Expr = Expr1 + Expr2.


ask_for_avail(Buyer, Product, ExprProduct, Seller):- consider_buying(Buyer, Product, ExprProduct
),
        can_offer(Seller, Product),
        ExprProduct>=1.
order(Buyer,Product,ExprProduct,Seller):- got_avail(Buyer,Product,ExprProduct,Seller,ExprAvl),


         ExprAvl >= ExprProduct.
ask_price(Buyer,Product,ExprProduct,Seller):-
got_avail(Buyer,Product,ExprProduct,ExprAvl,Seller),


                    ExprAvl >= ExprProduct.
pay(Buyer,Product, ExprProduct, Seller, ExprPrice) :-
send_price(Seller,Product,ExprProduct,Buyer,ExprPrice), money(Buyer,ExprMoney), ExprMoney
>= ExprPrice, !.


%seller


got_avail(Buyer, Product, ExprProduct, Seller, ExprAvl):-
ask_for_avail(Buyer,Product,ExprProduct,Seller),


                                available_products(Seller,Product,ExprAvl).
available_products(Seller, Product,ExprAvl):-can_offer(Seller, Product),


                                                stock(Seller, Product, ExprAvl).
send_price(Seller,Product,ExprProduct,Buyer,ExprPrice) :-
order(Buyer,Product,ExprProduct,Seller),cost(Product,ExprCost),ExprPrice is ExprProduct *
ExprCost.
```

Βιβλιογραφία

[1] Georgios K. Giannikis and Aspassia Daskalopoulou: Assumption-based Reasoning in Dynamic Normative Agent Systems

[2] Aspassia Daskalopoulou and Georgios K. Giannikis , University of Thessaly,Greece: Autonomous hypothetical reasoning: the case for open-minded agents

[3] http://ai.uom.gr/Courses/AdvancedArtificialIntelligence/Slides/Prolog.pdf , Γιάννης Ρεφανίδης , ΠΑΝΕΠΙΣΤΗΜΙΟ ΜΑΚΕΔΟΝΙΑΣ,ΠΜΣΕ ΣΤΗΝ ΕΦΑΡΜΟΣΜΕΝΗ ΠΛΗΡΟΦΟΡΙΚΗ

[4] http://en.wikipedia.org

[5] http://cgi.di.uoa.gr/~ys02/lectures/propositional1spp.pdf

[6] http://lpis.csd.auth.gr/mtpx/agents/ – Michael Wooldridge, "An Introduction to MultiAgent Systems - Second Edition", John Wiley & Sons, ISBN: 978-0470519462, May 2009.

[7] M.Gelfond and V. Lifschitz. The stable model semantics for logic programming. pages 1070-1080, 1988

[8] M.Gelfond and V. Lifschitz. Classical negation in logic programs and disjunctive databases. New Generation Comput., 9(3/4): 365-386, 1991

[9] D. Nute. Defeasible Logic. In D. Gabbay, C.J. Hogger, and J.A. Robinson editors, *Handbook of Logic in Artificial Intelligence and Logic Programming, Nonmonotonic Reasoning and Uncertain Reasoning,* volume 3, pages 353-395. Oxford University Press 1994

[10] Intelligent agents: theory and practice, MICHAEL WOOLDRINGE and NICHOLAS R. JENNINGS, the Knowledge Engineering Review, Vol. 10:2, 1995, 115-152

[11] Ανάπτυξη Εφαρμογών με Σχεσιακές Βάσεις Δεδομένων, Δρ. Θεοδώρου Παύλος, http://www.aeahk.gr/anagnwsthrio/sxesiakes%20baseis%20dedomenwn.pdf

[12] Hunter, Anthony. "Using default logic for lexical knowledge." *Qualitative and Quantitative Practical Reasoning*. Springer Berlin Heidelberg, 1997. 322-335.

[13] Hunter, Anthony. "Intelligent text handling using default logic." *Tools with Artificial Intelligence, 1996., Proceedings Eighth IEEE International Conference on*. IEEE, 1996.

[14] Contextual Default Reasoning. Gerhard Brewka. Universität Leipzig. Augustusplatz 10-11. 04109 Leipzig, Germany brewka@informatik.uni-leipzig.de