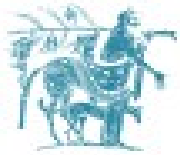


Πανεπιστήμιο Θεσσαλίας
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΗΛΕΚΤΡΟΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Τίτλος

"Υποθετικός συλλογισμός με αναπαραστάσεις αναιρέσιμης λογικής"

Title

"Assumption-based reasoning using defeasible reasoning representations"

Διπλωματική Εργασία
της
Σκρέκα Λαμπρινής

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 αυτής της πτυχιακής- για την καθοδήγησή της, τις πολύτιμες συμβουλές της και την στήριξή της, όπως επίσης και τον κύριο Ακρίτα Αλκιβιάδη - αναπληρωτή καθηγητή και συνεπιβλέπων της πτυχιακής.

Ένα μεγάλο ευχαριστώ στην οικογένεια μου, για την αγάπη τους, την υπομονή τους και την στήριξή τους σε εμένα όλα αυτά τα χρόνια.

Τέλος οι φίλοι, είναι πολύ σημαντικό κομμάτι στη ζωή μας και χωρίς αυτούς δε θα έκανα τίποτα από όλα αυτά και ακόμη και αν τα έκανα δεν θα είχαν κανένα απολύτως νόημα. Είναι εκεί πάντα για να μου θυμίζουν τι έχει πραγματική αξία και τους ευχαριστώ γι' αυτό.

Βόλος, Ιούνιος 2013

Περίληψη

Στην παρούσα εργασία πραγματευόμαστε την υλοποίηση και την ανάπτυξη της συλλογιστικής πορείας ενός πράκτορα, που λαμβάνει αποφάσεις ακολουθώντας κανόνες γραμματικής τύπου Defeasible Logic (Donald Nute).

Παρουσιάζουμε μία υπολογιστική τεχνική για τον τρόπο λήψης αποφάσεων ενός πράκτορα, βασιζόμενος σε αναιρέσιμη λογική. Ο πράκτορας, ανάλογα με την τρέχουσα γνώση σε κάθε δεδομένη χρονική στιγμή, εντοπίζει τις κατάλληλες υποθέσεις και δρα σύμφωνα με αυτές, ώστε να εξάγει συμπεράσματα. Όταν τα συμπεράσματα αυτά αναιρεθούν, οι πράκτορες μπορούν να ανακατασκευάσουν την άποψή τους για τον περιβάλλον στο οποίο βρίσκονται.

Στόχος μας λοιπόν είναι, για έναν πράκτορα ο οποίος δρα βάσει αναιρέσιμης λογικής να εξάγει συμπεράσματα ακόμα και όταν δεν έχει γνώση για το περιβάλλον του. Αυτό γίνεται μέσω λήψης υποθέσεων. Για τις περιπτώσεις όπου ο πράκτορας αντιμετωπίζει σύγκρουση κανόνων, ορίζουμε έναν μηχανισμό για την επίλυση αυτών.

Abstract

In the present paper we discuss the implementation and development of an agent reasoning scheme which is based on Defeasible Logic (Donald Nute).

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.

In particular, when an agent acts based on defeasible logic, we intent to draw conclusions even if he has insufficient knowledge of his environment. This is accomplished by making assumptions which however may increase the cases of conflicts. Conflicts occur when rules which disprove each other hold simultaneously. In order to deal with this inherent issue of Defeasible Logic we have devised a mechanism which eliminates such conflicts.

Πίνακας περιεχομένων

1

Εισαγωγή

1.1 Αντικείμενο διπλωματικής

Στην καθημερινότητα μας ερχόμαστε αντιμέτωποι με καταστάσεις για τις οποίες δεν έχουμε πληροφορία για το περιβάλλον, έτσι αναγκαζόμαστε να φτάνουμε σε συμπεράσματα μέσω υποθέσεων. Στην τυπική λογική, ο ανοιχτός σε υποθέσεις κόσμος (open world assumption OWA), είναι η παραδοχή ότι κάτι μπορεί να ισχύει ανεξάρτητα από το εάν είναι γνωστό ή όχι από οποιοδήποτε μεμονωμένο παρατηρητή ή πράκτορα. Είναι το αντίθετο του κλειστού σε υποθέσεις κόσμου (CWA), στον οποίο κάθε δήλωση που δεν είναι γνωστή για το αν είναι αληθινή είναι ψευδής [7].

Η αναγκαιότητα του να λαμβάνουμε απόφαση μέσω υποθέσεων, όταν δεν έχουμε αρκετή πληροφορία και να μη θεωρούμε την κατάσταση αυτή ρητά ψευδής, είναι επιτακτική. Για παράδειγμα, εάν γνωρίζουμε ότι η Μαρί είναι πολίτης της Γαλλίας και για τον Γιάννη δεν έχουμε καμία πληροφορία για την καταγωγή του, θα πρέπει να μπορούμε να υποθέσουμε και γι' αυτόν ότι είναι από τη Γαλλία και όχι να δηλώνουμε ρητά ότι δεν είναι. Ένας πράκτορας επιδιώκει πάντα να βγάλει συμπέρασμα η έλλειψη πληροφορίας μπορεί να αποτελέσει τροχοπέδη στην επίτευξη του στόχου αυτού. Οι υποθέσεις εξυπηρετούν την επίτευξη του στόχου έστω και αν αργότερα έρθουν δεδομένα που τις αναιρούν. Σαφέστατα, η χρήση υποθέσεων κάνει ένα πρόγραμμα πιο ελκυστικό και ευέλικτο και ξεφεύγει από την στυγνότητα και αδιαλλακτικότητα προγραμμάτων του CWA που εξάγουν θετικά συμπεράσματα μόνο με επαρκή γνώση.

Έτσι η εργασία αυτή έχει ως στόχο την ανάπτυξη μίας βιβλιοθήκης μέσω της οποίας

προσπαθούμε να προσφέρουμε την δυνατότητα υποστήριξης υποθέσεων σε γραμματική τύπου Defeasible Logic, καθώς και την επίλυση συγκρούσεων, όπου αυτές προκύπτουν.

Τα συμπεράσματα που παίρνουμε όταν έχουμε έλλειψη πληροφορίας, δεν είναι πλέον αρνητικά, αλλά μέσω της λήψης υποθέσεων επιτυγχάνουμε έστω και προσωρινά αναιρέσιμα συμπεράσματα δίνοντας στον πράκτορα τη δυνατότητα να επιτύχει κάποια συμφωνία, η οποία όμως θα αναιρεθεί, αν εν τέλει λάβουμε γνώση αντίθετη με τις υποθέσεις μας. Συγκεκριμένα, ο πράκτορας μας πλέον, θεωρεί την έλλειψη γνώσης μίας κατάστασης ή ενός γεγονότος ως πιθανό θετικό συμπέρασμα εκτός κι αν υπάρχει ρητή δήλωση στο πρόγραμμα να μην γίνει υπόθεση για τον συγκεκριμένο κανόνα.

Επιπλέον, μέχρι τώρα στη γραμματική τύπου Defeasible Logic, ένας κανόνας ο οποίος αποδεικνυόταν με διαφορετικούς τρόπους έφτανε σε σύγκρουση και δεν μπορούσαμε να βγάλουμε κάποιο συμπέρασμα γι' αυτόν. Αυτό καταφέραμε να το επιλύσουμε δίνοντας προτεραιότητες.

Η εργασία αυτή επιδιώκει να αξιοποιήσει στο έπακρο τις δυνατότητες της Defeasible Logic καθώς και να επιλύσει οποιαδήποτε "προβλήματα", είτε αυτά είναι έλλειψη γνώσης είτε συγκρούσεις, τα οποία εμποδίζουν τον πράκτορα να βγάλει συμπεράσματα.

1.2 Πιθανές χρήσεις της Defeasible Logic

Μία εφαρμογή της Defeasible Logic είναι η αυτοματοποίηση των διαπραγματεύσεων μεταξύ πρακτόρων [8,9]. Κάτι που έχει ιδιαίτερη σημασία για συστήματα συναλλαγών, εφόσον οι πράκτορες είναι πολύ πιθανό να μην έχουν πλήρη γνώση του περιβάλλοντός τους λόγω της δυναμικής του φύσης. Έτσι η λήψη υποθέσεων βοηθά στην γεφύρωση του χάσματος που δημιουργείται εξαιτίας αυτής της έλλειψης πληροφορίας. Γενίκευση τέτοιου συστήματος μπορεί να θεωρηθεί και το παγκόσμιο διαδίκτυο (Web) [10]. Το οποίο μάλιστα αποτελείται τόσο από στατική γνώση όσο και από δυναμική. Με τον όρο στατική γνώση αναφερόμαστε στο περιεχόμενο των σελίδων που εμπεριέχουν γενικές αλήθειες και γεγονότα, ενώ η δυναμική φύση του προέρχεται από αλλαγές που εμπίπτουν στον τρόπο προβάλλονται αυτές οι πληροφορίες. Η Defeasible Logic μπορεί να εφαρμοστεί ώστε να καταλήξει κανείς σε συμπεράσματα, τα οποία βελτιώνουν την εμπειρία του τελικού χρήστη. Επιπρόσθετα, η Defeasible Logic βρίσκει εφαρμογή και στη θεωρία παιγνίων [11]. Η ικανότητα της να περιγράψει πιο περίπλοκους διαλόγους σε σχέση με την Default Logic καθώς είναι πιο ευέλικτη την καθιστά εξαιρετικό υποψήφιο για τέτοιου είδους συστήματα πρακτόρων.

Βέβαια η φύση της Defeasible Logic μπορεί να οδηγήσει έναν πράκτορα σε σύγκρουση γνώσης οπότε δεν θα ήταν ασφαλές για αυτόν να συμπεραίνει ένα αποτέλεσμα. Η δική μας υλοποίηση λύνει το παραπάνω πρόβλημα, αυξάνοντας την αξιοπιστία των απαντήσεων κάνοντας χρήση ενός μηχανισμού επίλυσης συγκρούσεων.

1.3 Οργάνωση κειμένου

Το κείμενο της διπλωματικής εργασίας παρουσιάζει την ακόλουθη διάρθρωση. Στο κεφάλαιο 2 παρουσιάζονται θέματα θεωρητικού υποβάθρου. Στο κεφάλαιο 3 προχωρούμε σε αναλυτική περιγραφή της γραμματικής τύπου Defeasible Logic ώστε να επιτευχθεί η πλήρης κατανόησή της. Συνεχίζοντας,στο κεφάλαιο 4 συζητούνται θέματα σχεδιασμού της βιβλιοθήκης και η παρουσίαση του τρόπου υλοποίησής της. Ο επίλογος στο κεφάλαιο 7 ολοκληρώνει τη διάρθρωση του κειμένου. Έπειτα θα βρούμε το *Παράρτημα Α* το οποίο περιέχει μια μικρή περιγραφή δυσνόητων όρων. Στο *Παράρτημα Β* θα βρούμε ένα σύντομο εγχειρίδιο χρήσης της πλατφόρμας SWI- Prolog που χρησιμοποιήσαμε για την ανάπτυξη της βιβλιοθήκης μας. Επίσης στο *Παράρτημα Γ* θα βρείτε ένα σενάριο buyer-seller. Τέλος παρουσιάζουμε τη βιβλιογραφία που απαιτήθηκε.

2

Θεωρητικό υπόβαθρο

2.1 Εισαγωγικά

Για την επίτευξη του στόχου μας χρειάστηκαν κάποιες βασικές γνώσεις τις οποίες θα αναφέρουμε και θα αναλύσουμε σε αυτό το κεφάλαιο. Αρχικά ήρθαμε σε κοντά με την έννοια του λογικού πράκτορα και των πολυπρακτορικών συστημάτων. Εν συνεχεία, για την υλοποίηση και την κατανόηση της εργασίας χρειάστηκε βασική γνώση των σχεσιακών βάσεων δεδομένων . Επιπλέον, υλοποιήσαμε την εφαρμογή μας πάνω στην πλατφόρμα εκτέλεσης και επικοινωνίας SWI-Prolog και γνωρίσαμε τη γλώσσα Prolog. Επίσης χρειάστηκε να αποκτηθεί πλήρης κατανόηση της defeasible logic και των κανόνων της, καθώς είναι βασισμένη σε αυτή η d-Prolog (defeasible prolog) μία επέκταση της γλώσσας Prolog. Στα πλαίσια της διπλωματικής μας εργασίας επαυξήσαμε την d-Prolog με σκοπό την επίλυση συγκρούσεων και διαχείριση ερωτήσεων με μεταβλητά ορίσματα.

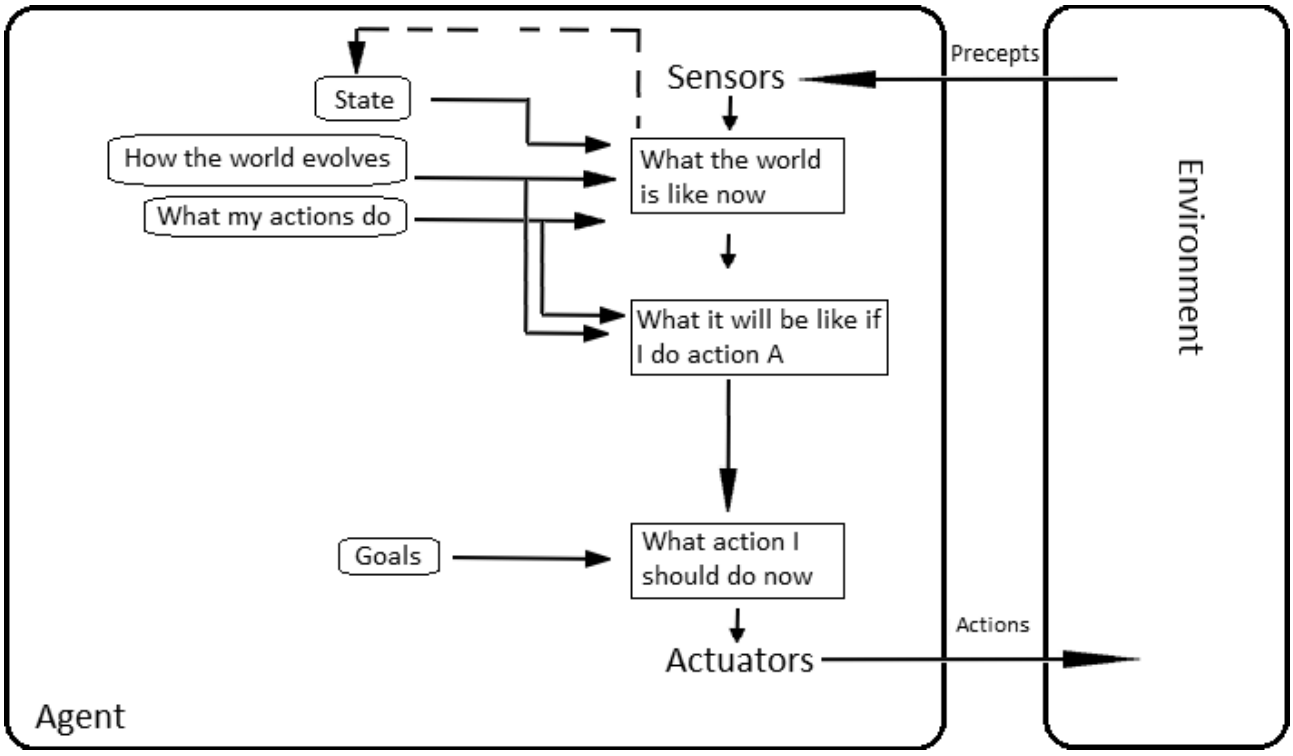
Παρακάτω ακολουθεί η ανάλυση των βασικών εννοιών που αναφέραμε πιο πάνω.

2.2 Η έννοια του πράκτορα λογισμικού

Ο πράκτορας είναι ένα υπολογιστικό σύστημα ικανό να δράσει ανεξάρτητα για λογαριασμό του χρήστη/ιδιοκτήτη του, δηλαδή μπορεί να αποφασίσει τι πρέπει να κάνει για να ικανοποιήσει τους στόχους για τους οποίους σχεδιάστηκε, χωρίς να χρειάζεται εντολές ανά πάσα στιγμή [3].

Οι πράκτορες θεωρούνται αυτόνομοι (ικανοί να αποφασίσουν για το τι ενέργειες πρέπει να κάνουν προκειμένου να ικανοποιήσουν τους σχεδιαστικούς στόχους τους) οπότε οι δομές συγχρονισμού και συντονισμού που χρησιμοποιούν δεν θεωρούνται προ-προγραμματισμένες. Αντ' αυτού χρησιμοποιούν μηχανισμούς συγχρονισμού και συντονισμού στη διάρκεια της λειτουργίας τους, δηλαδή δυναμικά.

Ο πράκτορας αλληλεπιδρά με το περιβάλλον ακολουθώντας το παρακάτω σχήμα, όπου παίρνει ερεθίσματα από το περιβάλλον, επεξεργάζεται το πως είναι ο κόσμος, πως οι κινήσεις θα τον μεταβάλλουν και τέλος επιλέγει την κατάλληλη ενέργεια για την επίτευξη του στόχου.



Εικόνα 1: Πράκτορας Λογισμικού

2.3 Η γλώσσα της Prolog

Η Prolog είναι μια γλώσσα λογικού προγραμματισμού γενικής χρήσης που κυρίως χρησιμοποιείται στον τομέα της τεχνητής νοημοσύνης. Στην Prolog, ο χρήστης δεν χρειάζεται να γράψει τι πρέπει να κάνει ο υπολογιστής γραμμή προς γραμμή, όπως συμβαίνει σε διαδικαστικές γλώσσες όπως η C και η Java [4]. Η γενική ιδέα πίσω από δηλωτικές γλώσσες είναι ότι θα περιγράψει την παρούσα κατάσταση (database and clauses). Το λογικό πρόγραμμα εκφράζεται με σχεσιακούς όρους 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

2.3.1 Τύποι δεδομένων στην Prolog

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

- 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.
- Σύνθετοι όροι: γράφονται ως “functor”, ακολουθούμενοι από μία λίστα όρων, διαχωρισμένη με κόμματα. Σύνθετος όρος θεωρούνται τα: (“Maria”, 1821) , (Zelda[tom,jim])
- Λίστες: διατεταγμένη συλλογή όρων. Συμβολίζεται με αγκύλες, μέσα στις οποίες υπάρχουν όροι, που διαχωρίζονται με κόμματα. [1,2,3], [πράσινο,κόκκινο,μπλέ] καθώς και η κενή λίστα [].
- Strings: ακολουθία χαρακτήρων που περιβάλλονται από εισαγωγικά.

2.3.2 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”. 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: cat(tom). Από αυτή τη δήλωση αντιλαμβανόμαστε ότι ο tom είναι γάτα. Ο παραπάνω κανόνας είναι ισοδύναμος με το: cat(tom):- true.

2.3.3 Μεταβλητές

Κάθε όρισμα ενός κατηγορήματος ή σύνθετου όρου που αρχίζει με κεφαλαίο γράμμα, η Prolog το θεωρεί μεταβλητή [5]. Σε αντίθεση με άλλες γλώσσες προγραμματισμού οι μεταβλητές στη Prolog δεν έχουν τύπο. Όπως θα δούμε στο κεφάλαιο ελέγχου της Prolog, μία μεταβλητή μπορεί να ταυτιστεί με οποιοδήποτε άτομο, κατηγόρημα, σύνθετο όρο ή αριθμό. Δύο μεταβλητές που έχουν το ίδιο όνομα και ανήκουν σε διαφορετικούς κανόνες ή γεγονότα της βάσης δεδομένων δεν σχετίζονται μεταξύ τους. Από τη στιγμή που μία μεταβλητή πάρει μία τιμή μέσα σε έναν κανόνα τότε αυτή δεν μπορεί ν' αλλάξει, έτσι σε κάθε χρονική στιγμή μία μεταβλητή μπορεί να θεωρηθεί ότι είτε έχει πάρει τιμή (instantiated) είτε ότι είναι ελεύθερη (uninstantiated).

2.3.4 Ερωτήσεις στην Prolog

Πάμε τώρα να δείξουμε πως κάνουμε ερωτήσεις στην Prolog. Έχοντας το παραπάνω γεγονός cat(tom). Η ερώτηση που θα θέταμε είναι της μορφής:

```
?-cat(tom).
True

?-cat(X).
X=tom
```

Έτσι μας επιστρέφει την τιμή (όνομα της γάτας) του κανόνα.
Αν προσθέταμε έναν κανόνα, έστω:

```
animal(X):-cat(X).
```

και ρωτούσαμε:

```
?-animal(tom).
True

?- animal(jerry).
False

?-animalmal(X).
X=tom
```

2.3.5 Αναδρομικοί Κανόνες

Οι αναδρομικοί κανόνες ορίζονται με τη βοήθεια του εαυτού τους. Παρακάτω βλέπουμε ένα παράδειγμα για την κατανόηση της αναδρομής στην γλώσσα της Prolog[5],
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.
is_ancestor(A,B):- is_parent(A,B).
is_ancestor(A,B):- is_parent(A,C), is_ancestor(C,B).
Ο πρώτος από τους δύο παραπάνω κανόνες ονομάζεται τερματική συνθήκη του αναδρομικού κανόνα.

2.4 Βάσεις δεδομένων

2.4.1 Εισαγωγή στις Βάσεις δεδομένων

Οι βάσεις δεδομένων περιλαμβάνουν πολλά αρχεία, με στοιχεία που αναφέρονται σε σχετιζόμενα χαρακτηριστικά των ίδιων οντοτήτων ή στοιχείων για οντότητες οι οποίες, εξαιτίας της χωρική τους εγγύτητας ή της χωρικής τους σύνδεσης, απαιτείται να ενωθούν ή να ομαδοποιηθούν 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[6].

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.

Οι πρώτοι τύποι βάσεων δεδομένων χρησιμοποίησαν ιεραρχικά μοντέλα ταξινόμησης ή μοντέλα δικτύων (ιδιοκτήτης/μέλος). Η εξέλιξη στο σχεσιακό μοντέλο σχετίζεται με τη δυνατότητα συσχέτισης των πινάκων μέσω διακριτών πεδίων. Για το συσχετισμό δύο πινάκων αρκεί ένα κοινό πεδίο, χαρακτηριστικό που κάνει το μοντέλο εύκαμπτο.

2.4.2 Χαρακτηριστικά πίνακα εγγραφών

Η είσοδος παρουσιάζει ένα στοιχείο και δεν υπάρχουν επαναλαμβανόμενες ομάδες στοιχείων. Σε κάθε στήλη όλα τα πεδία είναι του ίδιου είδους και 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).

Η επιλογή της σχεσιακής βάσης δεδομένων έγινε βάσει της δομής τους είναι πολύ ευέλικτη. Επίσης μπορούν να ανταποκριθούν σε όλες τις ερωτήσεις που μπορούν να μορφοποιηθούν χρησιμοποιώντας κανόνες της άλγεβρας Boolean, μαθηματικές εντολές κλπ. Επιτρέπουν την αναζήτηση, συνδυασμό και σύγκριση διαφορετικών ειδών δεδομένων, η πρόσθεση και η αφαίρεση νέων στοιχείων είναι ιδιαίτερα εύκολη, αφού αυτό ουσιαστικά σημαίνει πρόσθεση ή αφαίρεση ενός

πίνακα. Τέλος μέσω κοινών πεδίων επιτυγχάνεται η διατύπωση ερωτήσεων αναφερόμενες σε διαφορετικούς σχεσιακούς πίνακες.

2.5 Πλατφόρμα SWI-Prolog

Η SWI-Prolog είναι μια εφαρμογή ανοιχτού κώδικα της γλώσσας προγραμματισμού Prolog. Διαθέτει ένα πλούσιο σύνολο χαρακτηριστικών και βιβλιοθηκών, για τον λογικό προγραμματισμό , για multithreading, unit testing και GUI. Η SWI-Prolog, μπορεί να τρέξει σε λειτουργικά Unix, Windows, Macintosh και Linux.

3

Defeasible Logic

3.1 Εισαγωγή στην Defeasible Logic

Η Defeasible Logic δημιουργήθηκε από Donald Nute με ιδιαίτερη εστίαση στην αποτελεσματικότητα και στην εφαρμογή. Είναι ένας απλός και αποτελεσματικός κανόνας βασισμένος στη μη-μονοτονική φόρμα. Η λογική αυτή έχει αναπτυχθεί και επεκταθεί, και διάφορες παραλλαγές έχουν προταθεί. Η κύρια διαίσθηση της λογικής είναι να είναι σε θέση να αντλήσει «πειστικά» συμπεράσματα από επιμέρους και μερικές φορές από αντικρουόμενες πληροφορίες. Τα συμπεράσματα μας είναι προσωρινά, καθώς ένα συμπέρασμα μπορεί να ανακληθεί όταν έχουμε νέα δεδομένα πληροφοριών[2].

3.2 Η αναγκαιότητα της Defeasible Logic

Πολύ συχνά φτάνουμε σε συμπεράσματα τα οποία αργότερα αναιρούμε όταν λάβουμε πληροφορίες οι οποίες τα αντικρούουν. Οι καινούριες πληροφορίες αναιρούν την προηγούμενη συλλογιστική μας. Από 1980, σημαντική έρευνα στο AI επικεντρώθηκε στο να μοντελοποιήσει αυτού του τύπου τη συλλογιστική[1].

Σύμφωνα λοιπόν με τον Nute οι άνθρωποι συχνά απορρίπτουμε παλιά συμπεράσματα βασιζόμενοι σε νέα στοιχεία, ακόμη και όταν τα παλιά συμπεράσματα ήταν δικαιολογημένα από τα στοιχεία που είχαμε κατά τη στιγμή που φτάσαμε σε αυτά. Πράγμα που κάνει αυτή τη συλλογιστική μη μονοτονική (*Παράρτημα Α*). Χρειαζόμαστε ένα σύστημα συλλογιστικής που μας επιτρέπει να λαμβάνουμε πιθανές αποφάσεις με λιγότερα αποδεικτικά στοιχεία. Για τον λόγο αυτό χρειαζόμαστε έναν μηχανισμό για τη διόρθωση αυτού του είδους συλλογιστικής υπό το φως των περαιτέρω στοιχείων.

Η απουσία πληροφοριών μπορεί μερικές φορές να είναι ένας θετικός λόγος ώστε να πιστέψουμε ότι κάτι ισχύει. Υπάρχει γάλα στο ψυγείο; Κοιτάμε αλλά δε βλέπουμε γάλα. Η αποτυχία να βρούμε αποδείξεις για την ύπαρξη γάλακτος στην περίπτωση αυτή είναι ένας καλός λόγος να πιστεύουμε ότι δεν υπάρχει γάλα στο ψυγείο. Σε ένα άλλο παράδειγμα, πιστεύω ότι υπάρχει μια γάτα μπροστά μου. Το πιστεύω αυτό καθώς φαίνεται να υπάρχει μια γάτα μπροστά μου. Αυτό το στοιχείο είναι αρκετό για υποστηρίξει την ύπαρξη της γάτας μπροστά μου. Φυσικά, μπορούμε να σκεφτούμε καταστάσεις όπου θα ήταν λάθος η πεποίθησή μας. Μπορεί να έχω παραισθήσεις, ή θα μπορούσε να υπάρχει ένα ολόγραμμα μιας γάτας, ή θα μπορούσε να υπάρχει ένας καθρέφτης και η γάτα που νομίζω ότι βλέπω μπροστά μου είναι στην πραγματικότητα πίσω μου. Αλλά δεν έχω κανένα λόγο να πιστεύω ότι έχω παραισθήσεις και δεν υπάρχουν ενδείξεις ούτε προβολέα για ολογράμματα ή ούτε καθρέφτη. Η απουσία στοιχείων που θα αποδείκνυαν ότι η αντίληψή μου είναι λανθασμένη παρέχει μέρος της αιτιολόγησης για την πεποίθησή μου ότι υπάρχει μια γάτα μπροστά μου. Αν ήξερα ότι ήμουν σε μια κατάσταση όπου ολογραφικές εικόνες ήταν πιθανές, μπορεί να μην επέμενα στην πεποίθηση ότι υπάρχει μια γάτα μπροστά μου. Αυτό δεν είναι συλλογιστική με αβεβαιότητα με την κανονική έννοια. Είναι η αναγνώριση ότι η έλλειψη πληροφορίας μπορεί να βοηθήσει στη δικαιολόγηση των πεποιθήσεών μας.

3.3 Η γλώσσα της Defeasible Logic

Ο Nute ορίζει τον συμβολισμό $\vdash$ ως την κατάληξη της σχέσης κάποιου συστήματος Σ. Τότε το Σ είναι μονοτονικό μόνο στην περίπτωση για τα ζευγάρια S και T των τύπων και για κάθε τύπο φ της γλώσσας του Σ, if $T \vdash \varphi$, then $(T \cup S) \vdash \varphi$. Σημειώσαμε νωρίτερα ότι οποιοδήποτε συλλογιστικό σύστημα διατηρεί την αλήθεια πρέπει να είναι μονοτονικό, αλλά ένα συλλογιστικό σύστημα που διατηρεί τις δικαιολογήσεις δε θα είναι μονοτονικό. Αυτό σημαίνει ότι η πεποίθηση ότι το φ ίσως να μπορεί να δικαιολογηθεί με βάση την πίστη μας σε κάποιο άλλο σύνολο προτάσεων S. Αλλά θα μπορούσε να υπάρξει ένα σύνολο προτάσεων T τέτοιο ώστε, εάν φτάναμε να πιστεύουμε όλες τις προτάσεις στο $S \cup T$, δεν θα μπορούσε να δικαιολογηθεί πλέον το γεγονός ότι πιστεύουμε ότι ισχύει το φ.

Τα Defeasible συστήματα χρησιμοποιούν κανόνες των οποίων οι συνέπειες μπορεί να μην είναι αποσπάσιμες ακόμα και όταν οι προγονοί τους είναι υπολογισμένοι. Ένας Defeasible κανόνας μπορεί να ηττηθεί από ένα γεγονός ενός άλλου κανόνα. Ο δεύτερος κανόνας μπορεί είτε να αντικρούσει τον πρώτο κανόνα υποστηρίζοντας ένα επακόλουθο που τον αντικρούει είτε απλά αποκόβοντας τον πρώτο κανόνα υποδεικνύοντας μια κατάσταση στην οποία ο κανόνας δεν εφαρμόζεται. Η γνώση στην Defeasible Logic είναι οργανωμένη στα γεγονότα, στους κανόνες και στη σχέση υπεροχής.

Τα γεγονότα είναι αδιαμφισβήτητες δηλώσεις. Οι κανόνες της Defeasible Logic χωρίζονται σε strict κανόνες, defeasible κανόνες και undercutting defeaters. Οι κανόνες είναι μια κατηγορία εκφράσεων διακριτή από τους τύπους. Κατασκευάζονται χρησιμοποιώντας τρία σύμβολα $\rightarrow$, $\Rightarrow$ και $\rightsquigarrow$. Όπου $A \cup \{\varphi\}$ είναι ένα σύνολο τύπων. $A \rightarrow \varphi$ είναι ένας strict κανόνας, $A \Rightarrow \varphi$ είναι ένας defeasible κανόνας και $A \rightsquigarrow \varphi$ είναι ένας undercutting defeater. Σε κάθε περίπτωση, ο A ονομάζεται πρόγονος του κανόνα και το φ συνέπεια αυτού. Όπου $A = \{\psi\}$, στην ουσία ορίζεται ένα $A \rightarrow \varphi$ ως $\psi \rightarrow \varphi$, ομοίως για defeasible κανόνες και defeaters. Οι πρόγονοι για strict κανόνες και defeaters πρέπει να μην είναι κενοί ενώ οι πρόγονοι των defeasible κανόνων μπορεί να είναι κενοί. Ένας

κανόνας της μορφής $\varnothing \Rightarrow \varphi$ μπορεί να οριστεί πιο απλά ως $\Rightarrow \varphi$. Τέλος όσο επιτρέπονται οι μεταβλητές μέσα σε κανόνες, θα αντιμετωπίζονται οι κανόνες αυτοί, ως σχήματα για όλες τις δοθείσες τιμές.

Οι strict κανόνες δεν μπορούν ποτέ να ηττηθούν. Όχι μόνο δεν έχουν εξαιρέσεις, αλλά δεν θα μπορούσαν να έχουν εξαιρέσεις. Είναι η έκφραση της αναγκαίας σχέση μεταξύ προγόνων και επακόλουθων. Παραδείγματα αυτής της αξίωσης που θα παρουσιάζαμε με έναν strict κανόνα είναι «Οι εργένηδες δεν είναι παντρεμένοι" και "Οι πιγκουίνοι είναι πουλιά". Οι defeasible κανόνες αντιπροσωπεύουν ασθενέστερες συνδέσεις, οι οποίες μπορούν να ηττηθούν. Τέτοια παραδείγματα είναι “Οι πιγκουίνοι ζουν στην Ανταρκτική” και “Τα πτηνά πετούν”. Ένα παράδειγμα μία υπόθεσης είναι “Προφανώς, δεν υπάρχει ζωή στο φεγγάρι.” Οι defeaters είναι πολύ αδύναμοι για να υποστηρίξουν ένα συμπέρασμα. Ο ρόλος τους είναι να θέσουν υπό αμφισβήτηση ένα συμπέρασμα που θα μπορούσε αλλιώς να ισχύσει. Τέτοιου είδους επισημάνσεις τις ορίζουμε με το “μάλλον” για παράδειγμα “Ένα μουσκεμένο σπίρτο μάλλον δε θα καεί”.

Όπως αναφέρθηκε και πιο πάνω μια defeasible θεωρία περιλαμβάνει ένα αρχικό σύνολο γεγονότων και ένα σύνολο από κανόνες, αλλά θα πρέπει να περιλαμβάνει περισσότερα. Η σχέση υπεροχής σε μια δυαδική σχέση που ορίζεται πάνω στο σύνολο των κανόνων. Η σχέση υπεροχής καθορίζει τη σχετική δύναμη των δύο (αντικρουόμενων) κανόνων. Δύο κανόνες με επακόλουθα φ και ¬ φ συγκρούονται μεταξύ τους και θα μπορούσε να νικήσει το ένας το άλλο. Αναγνωρίζοντας πότε ακριβώς οι συνέπειες ενός συνόλου κανόνων θα οδηγήσει σε ασυνέπεια είναι ένα σοβαρό πρόβλημα, ιδίως για τη λογική first order όπου το ζήτημα δεν είναι σε θέση ακόμα να αποφασιστεί. Αυτό το πρόβλημα αντιμετωπίζεται ρητά ενσωματώνοντας την έννοια ενός συνόλου συγκρούσεων στις defeasible θεωρίες.

Κάθε σύνολο σύγκρουσης θα αποτελέσει ένα ελάχιστο σύνολο ασύμβατων τύπων, καθώς και ένα σύνολο των κανόνων σύγκρουσης, στην περίπτωση που οι συνέπειές τους αποτελούν ένα σύνολο συγκρούσεων. Είναι επιθυμητό το σύνολο των συγκρούσεων να τίθεται σε μια θεωρία που αντανακλά τις αναγκαίες σχέσεις που ενσωματώνονται στους strict κανόνες. If $\{\varphi, \psi\} \rightarrow \chi$ είναι στη θεωρία, τότε $\{\varphi, \psi, \neg\chi\}$ θα πρέπει να είναι ένα από σύνολα συγκρούσεων. Αυτή η ιδέα έχει ως σκοπό τα σύνολα των συγκρούσεων να “κλείνουν” με τους strict κανόνες της θεωρίας.

3.4 Η Συλλογιστική της Defeasible Logic

Η Defeasible Logic είναι μία “σκεπτικιστική” μη μονοτονική λογική, πράγμα που σημαίνει ότι δεν υποστηρίζει αντιφατικά συμπεράσματα. Αντ 'αυτού η Defeasible Logic επιδιώκει την επίλυση των συγκρούσεων[2]. Σε περιπτώσεις όπου υπάρχει κάποια υποστήριξη για το συμπέρασμα A αλλά και υποστήριξη για το συμπέρασμα ¬A, η Defeasible Logic δεν παίρνει ως συμπέρασμα αυθαίρετα μία από αυτές (εξού και το όνομα "σκεπτικιστική"). Εάν η υποστήριξη για το A έχει προτεραιότητα σε σχέση με την υποστήριξη για την ¬ A τότε συμπεραίνεται το A.

Τα συμπεράσματα μπορούν να ταξινομηθούν ως οριστικά ή αναιρέσιμα. Ένα οριστικό συμπέρασμα είναι ένα συμπέρασμα που δεν μπορεί να αποσυρθεί, όταν νέες πληροφορίες είναι διαθέσιμες. Ένα αναιρέσιμο συμπέρασμα είναι ένα προσωρινό συμπέρασμα και ενδέχεται να αποσυρθεί από τα νέα κομμάτια πληροφοριών. Επιπλέον, η λογική είναι σε θέση να πει εάν ένα συμπέρασμα είναι ή δεν είναι αποδείξιμο. Έτσι, είναι δυνατόν να έχουμε τους ακόλουθους 4 τύπους

συμπερασμάτων:

1. Θετικά οριστικά συμπεράσματα: πράγμα που σημαίνει ότι το συμπέρασμα αποδείχτηκε χρησιμοποιώντας μόνο γεγονότα και αυστηρούς κανόνες.
2. Αρνητικά οριστικά συμπεράσματα: πράγμα που σημαίνει ότι δεν είναι δυνατόν να αποδειχθεί το συμπέρασμα χρησιμοποιώντας μόνο γεγονότα και αυστηρούς κανόνες.
3. Θετικά αναιρέσιμα συμπεράσματα: πράγμα που σημαίνει ότι τα συμπεράσματα αποδείχτηκαν από αναιρέσιμους κανόνες
4. Αρνητικά αναιρέσιμα συμπεράσματα: πράγμα που σημαίνει ότι μπορεί κανείς να δείξει πως το συμπέρασμα αυτό δεν μπορεί να αποδειχτεί ούτε με αναιρέσιμους κανόνες.

3.5 Οι Προτεραιότητες της Defeasible Logic

Μία μέθοδος για τον καθορισμό των προτεραιοτήτων των κανόνων στις defeasible θεωρίες είναι να χρησιμοποιηθεί ειδικότητα. Ένας κανόνας με πρόγονο A λέγεται ότι είναι πιο συγκεκριμένος από έναν κανόνα με πρόγονο B, σε σχέση με μια θεωρία T, αν μπορεί να αντληθεί όλο το B από το A χρησιμοποιώντας μόνο κανόνες του T, αλλά όχι το αντίστροφο. Η σχέση υπεροχής σε μια θεωρία μπορεί να βασίζεται σε μια τέτοια έννοια ειδικότητας. Μια πιο ενδιαφέρουσα περίπτωση, είναι όταν υπάρχει ήδη κάποια ρητή σχέση με τους κανόνες μιας θεωρίας που χρησιμοποιείται ως ο πυρήνας της προτεραιότητας στη σχέση αυτή.

3.6 d-Prolog

Ο σχεδιασμός της Defeasible Logic έχει επηρεαστεί από θέματα εφαρμογής, και πιο συγκεκριμένα θέματα εφαρμογής στην γλώσσα Prolog. Η Defeasible Prolog είναι μία μη-μονοτονική επέκταση της Prolog η οποία πρόκειται για μία εφαρμογή της Defeasible Logic. Υπήρξαν διάφορες εκδόσεις της d-Prolog [1] που χρονολογείται από το 1986. Μερικά από τα χαρακτηριστικά της, συμπεριλαμβάνουν τις διαφορές που έχει η Defeasible Logic από την formal logic.

Ξεκινάμε δείχνοντας πώς ορίζεται τη γλώσσα της d-Prolog. Ένας μοναδιαίος τελεστής neg και δύο δυαδικοί infix functors := και :^ προστίθενται στην Prolog. Ο τελεστής neg είναι η άρνηση την οποία την ξεχωρίζουμε από την built-in άρνηση-από-αποτυχία (negation-by-failure, NBF) του τελεστή not ή \+. Η Atom είναι μια ατομική πρόταση, η Atom και η neg Atom είναι συμπληρωματικές η μία της άλλης. Η neg Atom μπορεί να εμφανιστεί είτε στο Head είτε στο Body ενός κανόνα. Προτάσεις της μορφής Head: = Body ονομάζονται defeasible κανόνες, και προτάσεις της μορφής Head:^ Body ονομάζονται undercutting defeaters ή απλά defeaters. Ένας defeasible κανόνας της μορφής Head: = true ονομάζεται υπόθεση. Αντίθετα, οι συνήθεις κανόνες της Prolog τους ονομάζουμε strict κανόνες.

Τα συμπεράσματα μπορούν να προέλθουν είτε αυστηρά είτε αναιρέσιμα. Ένα συμπέρασμα μπορεί να προέλθει αυστηρά, αν και μόνο αν μπορεί να προέλθει μόνο από γεγονότα και αυστηρούς κανόνες της βάση γνώσης μας. Μία πρόταση Goal θα προέλθει αυστηρά, μόνο στην περίπτωση που το ερώτημα *?- Goal.* επιτυγχάνει. Ένα από τα συμπεράσματα προέρχεται defeasibly(*Παράρτημα A*) φτάνοντας σε αυτό χρησιμοποιώντας όλες τις προτάσεις της βάσης γνώσης, συμπεριλαμβανομένων και των αναιρέσιμων κανόνων, υποθέσεις καθώς και defeaters.

Ένας μοναδιαίος functor @@ εισάγεται για να υποστηρίξει εξαντλητική έρευνα των ερωτημάτων. Χρειάζεται ένας τρόπος ώστε να αντλείται με ένα μόνο ερώτημα αν ισχύει ή όχι μια ατομική πρόταση και αν αυτό προήλθε είτε αυστηρά είτε αναιρέσιμα. Για την απάντηση στο ερώτημα *?- @@ Goal*, η d-Prolog θα δοκιμάσει όλες αυτές τις πιθανότητες και να δώσει μια κατάλληλη απάντηση: "σίγουρα ναι", "σίγουρα όχι", "μάλλον ναι", "μάλλον όχι", ή "δεν μπορώ να πω".

Εδώ είναι ένα παράδειγμα της d-Prolog βάσης γνώσης.

```
born_in(X,usa) :- born_in(X,atlanta).
neg born_in(X,usa) :=
native_speaker(X,greek).
born_in(stavros,atlanta) := true.
native_speaker(stavros,greek) := true.
```

Για αυτή τη βάση γνώσης, η d-Prolog απαντά στο ερώτημα *?- @@ born_in (stavros, USA)*. "προφανώς ναι", επειδή ο αυστηρός κανόνας έχει μεγαλύτερη προτεραιότητα από τον αναιρέσιμο κανόνα.

Ένα άλλο γνωστό παράδειγμα, το λεγόμενο Nixon Diamond, δείχνει πώς δύο defeasible κανόνες ίσως μπορούν να νικήσουν ο ένας τον άλλο. Το παράδειγμα αυτό σε d-Prolog, το βλέπουμε παρακάτω.

```
pacifist(X) := quaker(X).
neg pacifist(X) := republican(X).
quaker(nixon).
republican(nixon).
```

Η σωστή απάντηση στο ερώτημα:

```
?- @@ pacifist(nixon).
```

Είναι "δεν μπορώ να απαντήσω" δεδομένου ότι κανένας από τους δύο αναιρέσιμους κανόνες δεν υπερισχύει του άλλου και αμοιβαία καταρρίπτονται. Ωστόσο, θα μπορούσε να αποφασιστεί ότι σε αυτό το είδος συνθηκών, το "πολιτικό κόμμα" έχει προτεραιότητα έναντι στις "θρησκευτικές πεποιθήσεις". Για να το πετύχει αυτό, θα πρέπει να προστεθεί η παρακάτω πρόταση στην d-Prolog γνωσιακή βάση μας.

```
sup((pacifist(X) := quaker(X)),
(neg pacifist(X) := republican(X))).
```

Με την αυτή την προσθήκη το ερώτημα:

```
?- @@ pacifist(nixon).
```

Βγάζει ως απάντηση "μάλλον όχι".

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 Defeasible Logic, καθώς και την επίλυση συγκρούσεων, όπου αυτές προκύπτουν.

Ξεκινώντας από μία υπάρχουσα γραμματική και κάνοντας χρήση του τελεστή @@ ο χρήστης μπορεί να θέσει ερωτήσεις οι οποίες απαντώνται με ή χωρίς υποθέσεις. Οι τέσσερις πιθανές απαντήσεις είναι:

- “definitely yes”
- “definitely no”
- “presumably yes”
- “presumably no”

Για τις δύο τελευταίες απαντήσεις η έξοδος θα συμπεριλαμβάνει και τις υποθέσεις που έγιναν κατά την διερεύνηση της ερώτησης.

Ο τελεστής @@ γίνεται διαθέσιμος ενσωματώνοντας τη βιβλιοθήκη μας κάνοντας χρήση του *consult('dprolog_extended.pl')* στο αρχείο που περιλαμβάνει την αρχική γραμματική.

Κατά την ανάπτυξη της βιβλιοθήκης χρειάστηκε να κατανοήσουμε πλήρως τους strict και defeasible κανόνες και τους defeaters και να τους υλοποιήσουμε με τη γλώσσα της Prolog. Για δική μας ευκολία δεν χρειάστηκε να ξεκινήσουμε από μηδενική βάση καθώς η d-Prolog και οι κανόνες της υπάρχουν ήδη. Οι strict κανόνες έχουν τη μεγαλύτερη προτεραιότητα και ορίζονται όπως

προείπαμε με τον τελεστή :-. Οι defeasible κανόνες έχουν μικρότερη προτεραιότητα και ηττούνται από τους stict κανόνες και τους ορίζουμε με τον τελεστή :==, διότι ο τελεστής := που αναφέραμε πιο πάνω είναι built_in predicate στην prolog. Για τους δύο παραπάνω τύπους κανόνων υποστηρίζεται και η δήλωση άρνησής τους μέσω του μοναδιαίου τελεστή neg. Τέλος οι defeaters έχουν τον τελεστή :^.

Οι ερωτήσεις όπως είπαμε και πιο πάνω γίνονται με κάνοντας χρήση του μοναδιαίου τελεστή @@ στη μορφή @@ *Goal.*

4.1.1 Λήψη Υποθέσεων

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

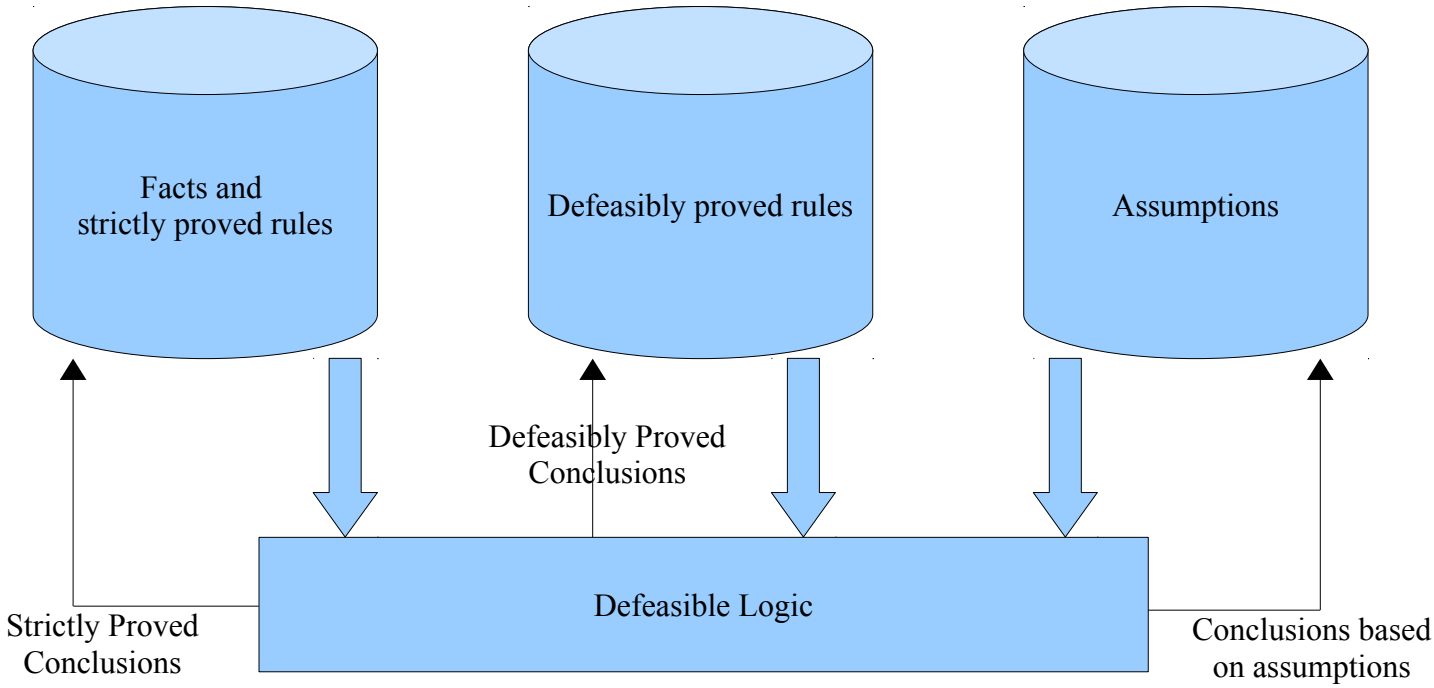
1. Η ερώτηση αφορά κανόνα, για τον οποίο δεν έχουμε γνώση, δηλαδή δεν υπάρχει στην αρχική γραμματική.
2. Η ερώτηση αφορά κανόνα, ο οποίος στο σώμα του βασίζεται σε απροσδιόριστες μεταβλητές. Δηλαδή μεταβλητές οι οποίες δεν έχουν λάβει τιμή, κάτι το οποίο είναι πιθανό όταν στην ροή του προγράμματος υπεισέρχονται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

Έτσι προκύπτει ο εξής διαχωρισμός συμπερασμάτων:

1. Τα γεγονότα και οι strict κανόνες οι οποίοι προκύπτουν χωρίς υποθέσεις και χωρίς να βασίζονται σε κάποιον defeasible κανόνα.
2. Τους defeasible κανόνες οι οποίοι αποδεικνύονται βασιζόμενοι χωρίς υποθέσεις είτε σε strict είτε σε άλλους defeasible κανόνες.
3. Τους strict ή defeasible κανόνες οι οποίοι προκύπτουν με τη βοήθεια υποθέσεων.

Τα συμπεράσματα αυτά αποθηκεύονται ανάλογα με το είδος τους και αργότερα ίσως να χρησιμοποιηθούν για την απόδειξη κάποιου άλλου κανόνα.

Η κατηγοριοποίηση των συμπερασμάτων σε γνωσιακές βάσεις και η ροή τους φαίνεται στο παρακάτω σχήμα:



Εικόνα 2: Βάσεις γνώσης στη Defeasible Logic

4.1.2 Επίλυση Συγκρούσεων

Ένα πρόβλημα που κληθήκαμε να λύσουμε ήταν αυτό των συγκρούσεων μεταξύ κανόνων που αναιρούνται μεταξύ τους. Το τελικό αποτέλεσμα των συγκρούσεων εμφανίζεται σε μορφή πίνακα παρακάτω:

Κανόνας\Ερώτηση	**Αντίπαλος Κανόνας**	**Έκβαση**
A :- B (B = strictly proved)	neg A :- C (C οτιδήποτε)	A :- B, άρα απάντηση definitely yes
A :== B (B οτιδήποτε)	neg A :- C (C strictly proved)	neg A :- C, άρα απάντηση definitely no
A :== B (B οτιδήποτε)	neg A :- C (C defeasible proved or with assumptions)	neg A :- C, άρα απάντηση presumably no
A :== B (B = defeasibly proved)	neg A :== C (C οτιδήποτε)	A :== B, άρα απάντηση presumably yes
A :== B (B = οτιδήποτε)	neg A :== C (C with assumptions)	A :== B, άρα απάντηση presumably yes
A :== B (B with assumptions)	neg A :== C (C without assumptions)	neg A :== C , άρα απάντηση presumably no

Ακολουθούμε την ίδια λογική για να επιλύσουμε τις συγκρούσεις που προκύπτουν κατά την διερεύνηση ερωτήσεων τύπου neg.

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('dprolog_extended.pl').* Αυτό είναι αρκετό για να χρησιμοποιήσει στην συνέχεια τον κανόνα *@@ Goal.* ο οποίος λειτουργεί σύμφωνα με τη λογική που περιγράψαμε στη *παράγραφο* 4*.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 Β*.

4.3 Λεπτομέρειες υλοποίησης

Ο τελεστής @@ αποφαίνεται για μία ερώτηση καταλήγοντας σε μία από τις τέσσερις πιθανές απαντήσεις που περιγράφουμε στην *παράγραφο 4.1.* Πιο συγκεκριμένα αν ένας κανόνας αποδεικνύεται strictly (*παράρτημα Α),* τότε η απάντηση είναι “definitely yes”. Ενώ για τους

υπόλοιπους κανόνες χρησιμοποιείται η λογική του μηχανισμού επίλυσης συγκρούσεων όπως την περιγράψαμε στην *παράγραφο 4.1.2.*

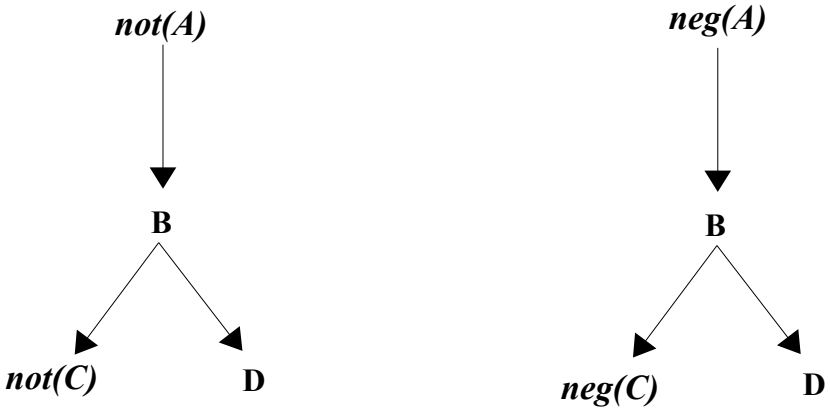
Για κάθε ερώτηση διατηρούνται τρεις βάσεις πληροφορίας. Η μία είναι αυτή που υπάρχει εκ σχεδιασμού από την Default Logic και περιέχει τους κανόνες οι οποίοι αποδεικνύονται κάνοντας χρήση γεγονότων και αυστηρά αποδεικνυόμενων κανόνων. Αυτή συμπεριλαμβάνει δηλαδή τις αρχικές δηλώσεις γνώσεως που υπάρχουν στη γραμματική του χρήστη καθώς και όλους τους κανόνες οι οποίοι προκύπτουν από αυτές. Ενώ οι κανόνες οι οποίοι βασίζονται σε defeasible rules ενσωματώνονται στην βάση γνώσης "Defeasibly Proved Rules". Τέλος διατηρείται μία βάση γνώσεως σε μορφή λίστας η οποία περιέχει τις υποθέσεις που παράχθηκαν κατά τη διερεύνηση ενός κανόνα.

Εσωτερικά ο τελεστής @@ χρησιμοποιεί τον κανόνα *def_der(Goal, J, Success),* για να αποδείξει όσες ερωτήσεις δεν προκύπτουν από strict κανόνες και γεγονότα. Πιο συγκεκριμένα μία ανάκληση αυτού του κανόνα για την ερώτηση *Goal* θα επιστρέψει το αν ισχύει η όχι στο *Success* και θα εισάγει στη λίστα *J* τις υποθέσεις που παρήχθησαν. Έτσι ένα *def_der(Goal, J, Success)* μπορεί να συμπεράνει ότι το *Goal* δεν μπορεί να αποφανθεί ως επιτυχές. Αυτό είναι απαραίτητο για να μπορέσουμε να προωθήσουμε όλες τις υποθέσεις που κρίθηκαν απαραίτητες για την έκβαση της ερώτησης ακόμα και αν συνείσφεραν αρνητικοί έλεγχοι κανόνων της μορφής *not(BodyComponent),* ή *neg(BodyComponent)*. Οι αντίστοιχοι ψευδοκώδικες είναι οι εξής:

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

και

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



Εικόνα 3: Αφηρημένα δένδρα

Στο σχήμα βλέπουμε το σκεπτικό μας για την διατήρηση των εμφωλευμένων *not/neg* στις υποθέσεις μας. Έτσι ξεκινώντας από το not(C)/neg(C), ελέγχουμε αν ισχύει το C. Αν ισχύει το C μπαίνει στη λίστα των υποθέσεων μας και το not(C)/neg(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)/neg(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.

Ο κανόνας *def_der(Goal, J, Success)* εσωτερικά χρησιμοποιεί τους κανόνες επίλυσης συγκρούσεων για να αποφασίσει την τιμή του *Success*. Αυτό γίνεται μέσω του κανόνα *rebutted(KB,Rule, Body, J)* ο οποίος ισχύει όταν ο κανόνας *Rule* ηττείται από κάποιον άλλο. Παρουσιάζουμε σε μορφή ψευδοκώδικα μία τέτοια περίπτωση.

```
% neg A :- B, και A :- C. Και τα δύο χωρίς υποθέσεις
rebutted(KB,Rule, Body, []) :-
     functor(Rule, 'neg', _),
     clause(Rule, Body),
     contrary(Rule, EnemyRule),
     clause(EnemyRule, Condition),
     Condition.
```

Η λογική με την οποία αντιμετωπίζουμε τις συγκρούσεις συνάδει με τον ορισμό της Defeasible Logic καθώς ουσιαστικά εφαρμόζει μία σχέση ανάδειξης προτεραιότητας στις περιπτώσεις όπου η d-Prolog δεν μπορούσε να βγάλει συμπέρασμα (“*can draw no conclusion”*). Οι βασικές αρχές για αυτή τη σχέση προτεραιότητας είναι:

1. Σε συγκρούσεις της ίδιας προτεραιότητας μορφής A και neg(A), προτεραιότητα δίνεται στη κατάφαση.
2. Σε συγκρούσεις όπου το neg(A) έχει μεγαλύτερη προτεραιότητα έναντι του A, υπερισχύει το neg(A).
3. Οι strict κανόνες έχουν πάντα μεγαλύτερη προτεραιότητα από τους defeasible ακόμα και

όταν βασίζονται σε υποθέσεις, το αποτέλεσμα τους όμως είναι "*presumably yes"/"presumably no"*.

4. Οι defeasible κανόνες χωρίς υποθέσεις έχουν μεγαλύτερη προτεραιότητα από τους defeasible με υποθέσεις. Το ίδιο ισχύει και για τους strict κανόνες.

4.4 Παραδείγματα τρεξίματος

Το "σενάριο" που θα παρουσιάσουμε είναι μεταξύ buyer – seller και υπάρχει στο *Παράρτημα Γ*.

Αν κάνουμε μία ερώτηση για την οποία γνωρίζουμε τα πάντα όπως:

```
?- @@pay(liza,nero,1,kurNikos,E).
   definitely, yes.
   E = 1.
```

Παίρνουμε ως απάντηση *"σίγουρα ναι"*.

Όπως επίσης στην περίπτωση της ερώτησης:

```
?- @@consider_buying(nikos,nero,1).
   definitely, no.
   true.
```

Παίρνουμε απάντηση *"σίγουρα όχι"*, καθώς ο Νίκος έχει νερό άρα δεν σκέφτεται να αγοράσει.

Σε μία ερώτηση για την οποία κάποια δεδομένα δεν είναι γνωστά για παράδειγμα:

```
?- @@pay(manos,nero,1,kurNikos,E).
   presumably, yes -
   and presumably, no - contradictory.
   Assumptions:
      [have_need(manos,dipsa),money(manos,_G141),_G141>=1*1,money(m
      anos,_G162),_G162>=1*1]
   presumably, yes.
   E = 1.
```

Σε αυτό το παράδειγμα για τον Μάνο δεν γνωρίζουμε αν έχει λεφτά ή ανάγκη για δίψα, έτσι θα γίνουν υποθέσεις ότι αυτά μπορεί να ισχύουν, αλλά και ότι αυτά μπορεί να μην ισχύουν, θα νικήσει σύμφωνα με τις προτεραιότητες που δώσαμε πιο πάνω, η απάντηση *"μάλλον ναι"*. Έτσι η λίστα των υποθέσεων περιλαμβάνει τα ενδεχόμενα για τα οποία δεν υπάρχει πλήρης γνώση.

Όταν όλα μας είναι άγνωστα όπως στην εξής ερώτηση:

```
?- @@pay(manos,tsai,1,giannis,E).
   Assumptions:
      [have_need(manos,_G134),satisfy_need(manos,_G134,tsai,1),cost
      (tsai,_G151),money(manos,_G154),_G154>=_G151*1,can_offer(kurG
      iannis,tsai),can_offer(kurGiannis,tsai),
      stock(kurGiannis,tsai,_G68), _G68>=1, cost(tsai,_G45), _G0 is
      1*_G45,cost(tsai,_G227),money(manos,_G230),_G230>=_G227*1]
    presumably, yes.
    True.
```

Παρατηρούμε ότι κάνει υποθέσεις για τα πάντα καθώς δεν έχει γνώση για τα χρήματα του buyer Μάνος, ούτε για το απόθεμα του seller Γιάννη, αλλά ούτε και για την τιμή του product τσάι. Τέλος, η απάντησή μας θα είναι *"μάλλον ναι"*.

Εδώ δίνουμε ένα σενάριο, στο οποίο έχουμε σύγκρουση η οποία επιλύεται σύμφωνα με τον τρόπο που δείξαμε παραπάνω.

```
?- @@have_money_for(lamprini,nero,1).
   definitely, yes -
   and definitely, no - contradictory.
   definitely, yes
   true.
```

Ξέρουμε ότι money(lamprini,1), cost(nero, 1) και οι κανόνες για το have_money_for είναι:

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

neg have_money_for(Buyer, Product, ExprProduct) :-
   cost(Product, ExprCost),
   money(Buyer, ExprMoney),
   ExprMoney =< ExprCost*ExprProduct.
```

Εφόσον ισχύουν και οι δύο κανόνες, προτεραιότητα έχει αυτός που βγάζει κατάφαση.

Τέλος, ένα άλλο σενάριο σύγκρουσης όπου δίνουμε προτεραιότητα στον strict κανόνα και όχι στον defeasible είναι το εξής:

```
?- @@pay(lamprini,kafe,1,kurNikos,E).
   definitely, no.
   True.
```

Όπως είπαμε και πιο πάνω money(lamprini,1) και cost(kafe, 4) οι κανόνες:

```
neg pay(Buyer,Product, ExprProduct, _, _) :-
     neg have_money_for(Buyer, Product, ExprProduct).

pay(Buyer,Product, _, Seller, _) :==
     send_price(Seller,Product,_,Buyer,_),
     money(Buyer,_).
```

Γι΄ αυτό, το αποτέλεσμά μας είναι *"σίγουρα όχι"*.

5

Επίλογος

5.1 Σύνοψη και συμπεράσματα

Στα πλαίσια της εργασίας μας δημιουργήσαμε ένα σύστημα λήψης αποφάσεων για πράκτορες που ενεργούν βάση της Defeasible Logic προσθέτοντας μηχανισμούς τόσο για τη λήψη υποθέσεων όσο και την επίλυση συγκρούσεων, αναβαθμίζοντας έτσι την Defeasible Logic.

Ένας πράκτορας πλέον, ανάλογα με την τρέχουσα γνώση σε κάθε δεδομένη χρονική στιγμή, εντοπίζει τις κατάλληλες υποθέσεις και δρα σύμφωνα με αυτές, ώστε να εξάγει συμπεράσματα. Όταν τα συμπεράσματα αυτά αναιρεθούν, οι πράκτορες μπορούν να ανακατασκευάσουν την άποψή τους για τον περιβάλλον στο οποίο βρίσκονται. Για τις περιπτώσεις όπου ο πράκτορας αντιμετωπίζει σύγκρουση κανόνων, ορίζουμε έναν μηχανισμό για την επίλυση αυτών, βάσει προτεραιοτήτων.

Κάνοντας αυτοαξιολόγηση της υλοποίησής μας εντοπίζουμε την ακόλουθη αδυναμία. Επειδή ο πράκτορας μας συμπεριλαμβάνει υποθέσεις στην βάση γνώσης του για τις περιπτώσεις όπου έχει άγνοια για το περιβάλλον του μπορεί να υποπέσει σε λογικά σφάλματα. Πιο συγκεκριμένα η λήψη μιας υπόθεσης ότι μία ενέργεια μάλλον ισχύει, ώστε να συνεχιστεί ο συλλογισμός, δεν είναι αξιόπιστη ενέργεια. Αυτό σημαίνει πως ο πράκτορας στην παρούσα κατάσταση δεν είναι ικανός για κρίσιμες λειτουργίες όπως για παράδειγμα έλεγχο εργοστασιακών μονάδων. Το πιο προφανές παράδειγμα λανθασμένης κρίσης είναι η αδυναμία του πράκτορα να εξάγει σωστό συμπέρασμα για λογικές εκφράσεις οι οποίες δεν είναι

απαραίτητο να είναι πλήρως ορισμένες. Έτσι για μία ερώτηση (G>1,G<1) η απάντηση που θα πάρουμε είναι “μάλλον ναι” , αποτέλεσμα το οποίο είναι σίγουρα λανθασμένο παρότι δεν υπάρχει γνώση για την τιμή του ορίσματος G.

5.2 Μελλοντικές Επεκτάσεις

Εντοπίζουμε την ανάγκη να γίνεται σωστή αποτίμηση των λογικών εκφράσεων ακόμα και όταν ένας πράκτορας λειτουργεί υπό συνθήκες ελλιπούς γνώσεως. Έτσι η δική μας πρόταση για μελλοντική εργασία είναι η ανάπτυξη ενός μηχανισμού ο οποίος θα εντοπίζει και θα επιλύει τέτοιου είδους λογικά σφάλματα όπως αυτά παρουσιάζονται στην *Παράγραφο 5.1*.

Συμπληρωματικά προτείνουμε την επέκταση της βιβλιοθήκης ώστε να υπάρχει πρόβλεψη για συστήματα στα οποία δρουν πράκτορες οι οποίοι εκτελούν διαφορετικές λειτουργίες. Σε αυτό το σημείο φέρνουμε ως παράδειγμα το σενάριο που χρησιμοποιήσαμε για να αξιολογήσουμε την επίδοση της βιβλιοθήκης μας. Μία μελλοντική επέκταση λοιπόν θα μπορούσε να λάβει υπόψιν της τη φύση του εκάστοτε πράκτορα αναλύοντας τους κανόνες βάση των οποίων δρα ώστε να καταλήγει σε συμπεράσματα με αποδοτικότερο τρόπο.

Παράρτημα Α

(1)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.

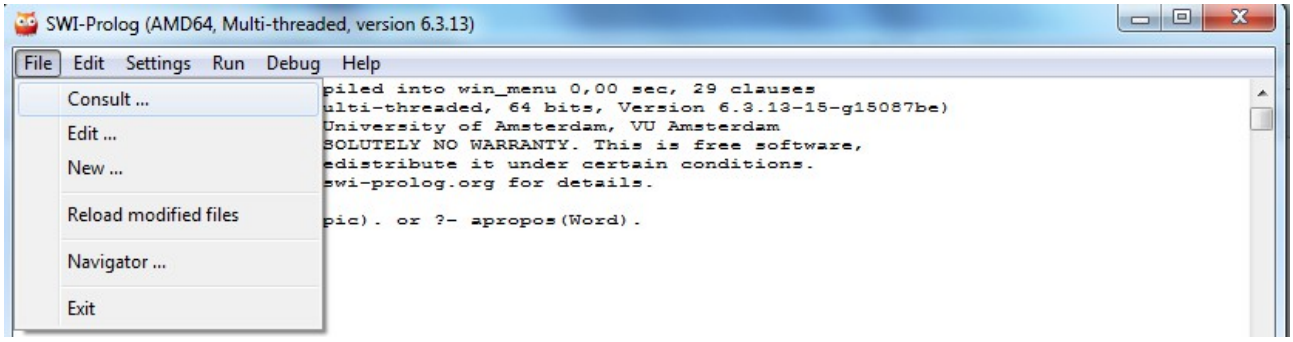
(2)Strictly, λέγοντας ότι κάτι αποδείχτηκε strictly εννοούμε ότι το συμπέρασμα αυτού του κανόνα πάρθηκε βάση γεγονότων και strict κανόνων, χωρίς καθόλου υποθέσεις.

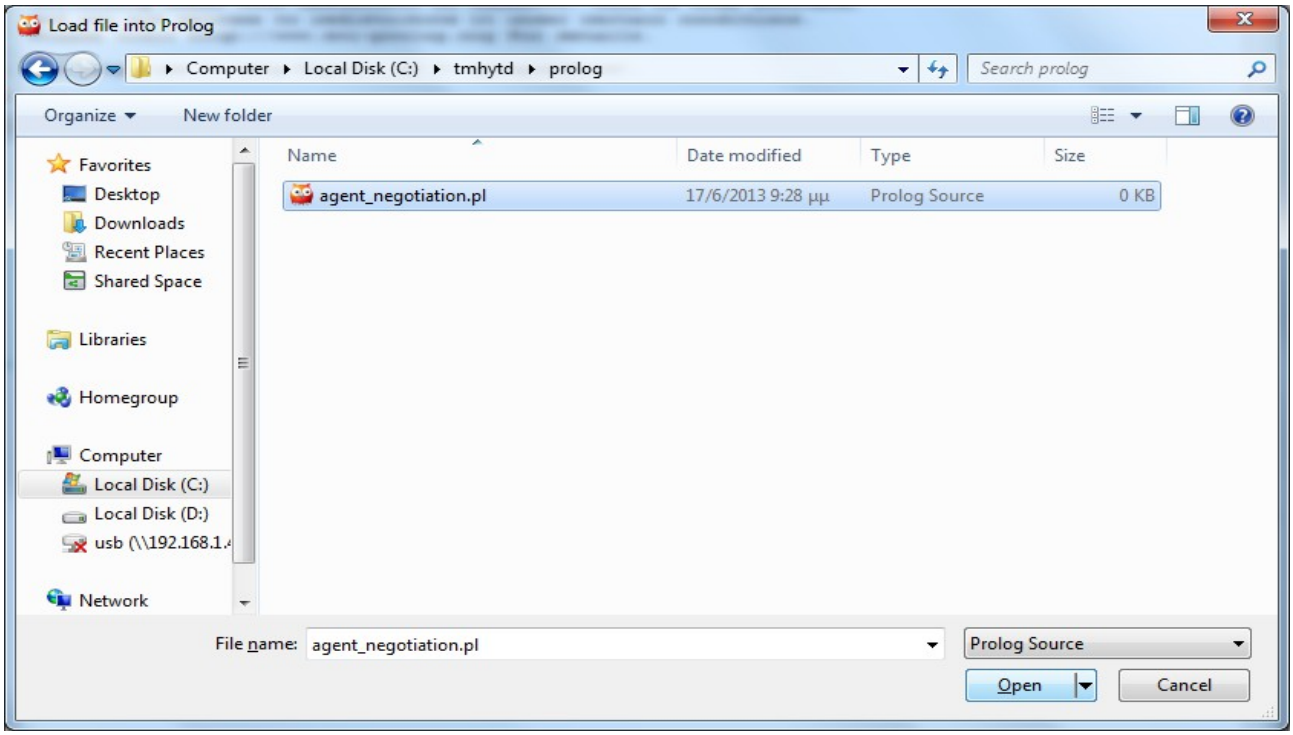
(3)Defeasibly, λέγοντας ότι κάτι αποδείχτηκε defeasibly, εννοούμε ότι το συμπέρασμα αυτού του κανόνα αποδείχτηκε με τη χρήση μόνο defeasible κανόνων, δηλαδή το αποτέλεσμα αυτό μπορεί να αναιρεθεί.

Παράρτημα Β

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 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.

```
:- consult('dprolog_extended.pl').
do_not_assume(have_at_least).

stock(kurNikos,nero,1000).
stock(kurNikos,kafe,500).
have_need(maria, dipsa).
have_need(lamprini, upnhlia).
have_need(lamprini, dipsa).
have_need(liza, dipsa).
satisfy_need(_, dipsa, nero, 1).
satisfy_need(_,upnhlia,kafe,1).
have_at_least(nikos, nero, 1).
money(nikos, 1000).
money(maria,0).
money(lamprini, 1).
money(liza,10).
cost(nero, 1).
cost(kafe, 4).
can_offer(kurNikos, nero).
can_offer(kurNikos,kafe).

%Buyer

neg missing(Buyer, Product, ExprProduct) :-
     have_at_least(Buyer, Product, ExprProduct).

missing(Buyer, Product, ExprProduct) :-
     have_need(Buyer, Need),
     satisfy_need(Buyer, Need, Product, ExprProduct).

consider_buying(Buyer, Product, ExprProduct) :-
     missing(Buyer, Product, ExprProduct),
     have_money_for(Buyer, Product, ExprProduct).

neg consider_buying(Buyer, Product, ExprProduct) :-
     neg(have_money_for(Buyer, Product, ExprProduct)).

neg consider_buying(Buyer, Product, ExprProduct) :==
     neg(missing(Buyer, Product, ExprProduct)).

have_money_for(Buyer, Product, ExprProduct) :-
     cost(Product, ExprCost),
     money(Buyer, ExprMoney),
     ExprMoney >= ExprCost*ExprProduct.

neg have_money_for(Buyer, Product, ExprProduct) :-
     cost(Product, ExprCost),
```

```
     money(Buyer, ExprMoney),
     ExprMoney =< ExprCost*ExprProduct.

ask_for_avail(Buyer, Product, ExprProduct, Seller):-
     consider_buying(Buyer, Product, ExprProduct ),
     can_offer(Seller, Product),
     ExprProduct>=1.
neg ask_for_avail(Buyer, Product, ExprProduct, _) :-
     neg(have_money_for(Buyer, Product, ExprProduct)).

neg ask_for_avail(Buyer, Product, ExprProduct, _):==
     neg(missing(Buyer, Product, ExprProduct)).

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

neg order(Buyer,Product,ExprProduct,Seller):-
     neg(got_avail(Buyer,Product,ExprProduct,Seller,_)).

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

neg ask_price(Buyer,Product,ExprProduct,Seller):-
     neg(got_avail(Buyer,Product,ExprProduct,Seller,_)).


pay(Buyer,Product, ExprProduct, Seller, ExprPrice) :-
     send_price(Seller,Product,ExprProduct,Buyer,ExprPrice),
     have_money_for(Buyer, Product, ExprProduct).

pay(Buyer,Product, _, Seller, _) :==
     send_price(Seller,Product,_,Buyer,_),
     money(Buyer,_).

neg pay(Buyer,Product, ExprProduct, Seller, ExprPrice) :-
     neg send_price(Seller,Product,ExprProduct,Buyer,ExprPrice).

neg pay(Buyer,Product, ExprProduct, _, _) :-
     neg have_money_for(Buyer, Product, ExprProduct).

% Seller

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

```
got_avail(Buyer, Product, ExprProduct, Seller, _ ) :==
     ask_for_avail(Buyer,Product,ExprProduct,Seller).

neg got_avail(Buyer, Product, ExprProduct, Seller, _ ) :-
     neg(ask_for_avail(Buyer,Product,ExprProduct,Seller)).

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.

neg send_price(Seller,Product,ExprProduct,Buyer,_) :-
     neg(order(Buyer,Product,ExprProduct,Seller)).

send_price(Seller,Product,ExprProduct,Buyer,_) :==
     order(Buyer,Product,ExprProduct,Seller).
```

Βιβλιογραφία

[1] Defeasible Logic, Donald Nute, Department of Philosophy and Artificial Intelligence Center The University of Georgia,Athens, GA 30605, U.S.A. dnute@uga.edu

[2] http://www.defeasible.net/

[3] http://inf-server.inf.uth.gr/courses/CE423/ Εισαγωγή στα Συστήματα Πρακτόρων και τους Νοήμονες Πράκτορες. – Wooldridge M. (2002). An Introduction to Multiagent Systems. John Wiley, Chichester. ISBN 047149691X

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

[5] http://www.icsd.aegean.gr/lecturers/stamatatos/courses/Logic/Prolog/Ch2/Ch2_1-2.htm
Ivan Bratko. Prolog Programming for Artificial Intelligence. Addison Wesley, 1994.
Leon Sterling and Ehud Shapiro. The Art of Prolog. The MIT Press, 1994.
Richard O'Keefe. The Craft of Prolog (Logic Programming). The MIT Press, 1990.
W. F. Clocksin and C. S. Melish. Programming in Prolog: Using the ISO Standard. Springer 5 edition, 2003.
Μ. Κατζουρακη, Μ. Γεργατσούλης και Σ. Κόκκοτος. Προγραμματίζοντας στη λογική . Ελληνική Εταιρία Επιστημόνων Η/Υ και Πληροφορικής , 1991.
Σπύρος Ξανθάκης. Prolog - Τεχνικές Προγραμματισμού. Εκδόσεις Νέων Τεχνολογιών, 1993.
Γεώργιος Μητακίδης. Από τη ΛΟΓΙΚΗ στο ΛΟΓΙΚΟ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟ και την Prolog. Εκδόσεις Καρδαμίτσα, 1992.

[6] http://el.wikipedia.org/wiki/%CE%A3%CF%87%CE%B5%CF%83%CE%B9%CE%B1%CE%BA%CE%AE_%CE%B2%CE%AC%CF%83%CE%B7_%CE%B4%CE%B5%CE%B4%CE%BF%CE%BC%CE%AD%CE%BD%CF%89%CE%BD – ISO/IEC 13211: Information technology — Programming languages — Prolog. International Organization for Standardization, Geneva.

[7] http://en.wikipedia.org/wiki/Open_world_assumption – *Russell*, *Stuart* J.; Norvig, Peter (2010). *Artificial Intelligence: A Modern Approach* (3rd ed.). Upper Saddle River: Prentice Hall. ISBN 9780136042594.

[8] Skylogiannis, Thomas, et al. "DR-NEGOTIATE–A system for automated agent negotiation with defeasible logic-based strategies." *Data & Knowledge Engineering* 63.2 (2007): 362-380.

[9] Antoniou, Grigoris, et al. "DR-BROKERING: A semantic brokering system."*Knowledge-Based Systems* 20.1 (2007): 61-72.

[10] Antoniou, Grigoris, and Antonis Bikakis. "DR-Prolog: A system for defeasible reasoning with rules and ontologies on the semantic web." *Knowledge and Data Engineering, IEEE Transactions on* 19.2 (2007): 233-245.

[11] Roth, Bram, et al. "Strategic argumentation: a game theoretical investigation."*Proceedings of the 11th international conference on Artificial intelligence and law*. ACM, 2007.