
ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ


# Σημασιολογική Ταξινόμηση Δεδομένων

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του


**ΚΩΝΣΤΑΝΤΙΝΟΥ Π. ΡΩΜΕΣΗ**



**Επιβλέπων :** Γιώργος Στάμου
Επίκουρος Καθηγητής Ε.Μ.Π.



Αθήνα, Μάρτιος 2017

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

# Σημασιολογική Ταξινόμηση Δεδομένων

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

**ΚΩΝΣΤΑΝΤΙΝΟΥ Π. ΡΩΜΕΣΗ**

**Επιβλέπων :** Γιώργος Στάμου
Επίκουρος Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 23η Μαρτίου 2017.

| | | |
|---|---|---|
| ..... | ..... | ..... |
| Γιώργος Στάμου<br>Επίκουρος Καθηγητής Ε.Μ.Π. | Ανδρέας-Γεώργιος Σταφυλοπάτης<br>Καθηγητής Ε.Μ.Π. | Στέφανος Κόλλιας<br>Καθηγητής Ε.Μ.Π. |

Αθήνα, Μάρτιος 2017

.....

**Κωνσταντίνος Π. Ρωμέσης**

Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

# Περίληψη

Ένα πεδίο έντονου ερευνητικού ενδιαφέροντος είναι η αναπαράσταση και διαχείριση της γνώσης στα υπολογιστικά συστήματα. Η επίτευξη των στόχων της ερευνητικής κοινότητας θα οδηγήσει στη δημιουργία νέων επαναστατικών εφαρμογών σε τομείς όπως η πρόσβαση σε δεδομένα και η διαλειτουργικότητα συστημάτων. Σημαίνοντα ρόλο στην υλοποίηση των βάσεων γνώσης, που απαιτούνται από αυτές τις εφαρμογές, παίζουν οι Περιγραφικές Λογικές. Μία πολλά υποσχόμενη τεχνική διαχείρισης εκτεταμένων βάσεων γνώσης είναι η μεταγραφή των ερωτημάτων που απευθύνονται σε αυτές.

Αντικείμενο της παρούσας διπλωματικής εργασίας ήταν η ανάπτυξη μιας εφαρμογής που θα επιτρέπει την αποτίμηση μεταγραφών datalog, που απαιτούνται από βάσεις γνώσης με την εκφραστικότητα περιγραφικών λογικών όπως η ELHI. Η σχεδίαση της εφαρμογής έγινε με στόχο την ευελιξία στη χρήση σε διαφορετικά περιβάλλοντα και την εύκολη επεκτασιμότητα. Υλοποιεί μια σειρά βελτιστοποιήσεων στη διαδικασία αποτίμησης. Κάποιες είναι γνωστές στο πεδίο των επαγωγικών βάσεων δεδομένων ενώ άλλες ανταποκρίνονται στις ιδιαίτερες απαιτήσεις που υπάρχουν από την εφαρμογή. Υποστηρίζονται βάσεις δεδομένων SQL και RDF.

**Λέξεις Κλειδιά:** Βάσεις γνώσης, Περιγραφικές Λογικές, Μεταγραφή Ερωτημάτων, Datalog, DL-lite, ELHI, Rapid, RDF, SQL.

# Abstract

Knowledge representation and management in computational systems is a field of intense research interest. The realization of the goals of the research community will lead to the creation of new and revolutionary applications in areas such as data access and system interoperability. Description Logics play a seminal role in the implementation of the knowledge bases required by such applications. A promising technique for the management of extended knowledge bases is the rewriting of the queries that are posed to them.

The object of this thesis was the development of an application that will allow the evaluation of datalog rewritings that are required by knowledge bases with the expressivity of description logics such as ELHI. The application was designed with the goal of flexibility of use in different environments and ease of extendibility. It implements a series of optimizations in the evaluation process. Some are known from the field of deductive databases while others respond to the specific demands posed to the application. RDF and SQL databases are supported.



**Keywords:** Knowledge Bases, Description Logics, Query Rewriting, Datalog, DL-lite, ELHI, Rapid, RDF, SQL.

# Ευχαριστίες

Θα ήθελα να ευχαριστήσω την οικογένειά μου για την συνεχή υποστήριξη, συμπαράσταση και κατανόησή τους. Τον Δρ. Γιώργο Στάμου επίκουρο καθηγητή Ε.Μ.Π. για το ενδιαφέρον και την εμπιστοσύνη που έδειξε στο πρόσωπό μου και τον ερευνητή Δρ. Αλέξανδρο Χορταρά για τις καθοριστικές συμβουλές του.

# Πίνακας περιεχομένων

# 1

# *Εισαγωγή*

Μια λειτουργία που γίνεται όλο και πιο απαραίτητη σε πληθώρα πρακτικών εφαρμογών είναι η σημασιολογική ταξινόμηση δεδομένων. Η συνεχώς αυξανόμενη δυνατότητα αποθήκευσης και διαχείρισης μεγάλου όγκου πληροφορίας σε βάσεις δεδομένων, αποτέλεσε μια από τις κύριες κινητήριες δυνάμεις πίσω από την εδραίωση της επιστήμης των υπολογιστών ως ενός από τα σημαντικότερα εργαλεία για την επίτευξη προόδου σχεδόν σε όλους τους τομείς της ανθρώπινης δραστηριότητας. Για δεκαετίες οι δυνατότητες των βάσεων συνέχισαν να εξελίσσονται καλύπτοντας όλο και μεγαλύτερο εύρος χρηστών και εφαρμογών. Η πρόοδος στο λογισμικό και το υλικό επέτρεψε την αποθήκευση όλο και μεγαλύτερου όγκου δεδομένων και την εξυπηρέτηση σύνθετων ερωτημάτων ενώ παράλληλα οι χρόνοι απόκρισης βελτιώνονταν. Κατέληξαν συνεπώς να χρησιμοποιούνται σε ένα ιδιαίτερα ευρύ φάσμα εφαρμογών. Σε αντίθεση, ωστόσο, με τις τεχνικές τους δυνατότητες, η συνταγή της αποθήκευσης και ανάκτησης δεδομένων διατηρούσε τα ίδια γενικά χαρακτηριστικά.

Σήμερα η επέκταση των δυνατοτήτων των βάσεων δεδομένων εντός των κλασσικών σεναρίων χρήσης, ενώ συνεχίζει φυσικά να είναι ιδιαίτερα ευπρόσδεκτη, αρχίζει να δείχνει μη επαρκής. Η ευρύτατη εξάπλωση των συστημάτων διαχείρισης βάσεων δεδομένων που με τη μία ή την άλλη μορφή πλέον βρίσκονται πίσω από τις περισσότερες εφαρμογές οδήγησε σε ένα νέο τύπο προβλήματος. Η ανάγκη ενσωμάτωσης δεδομένων από ετερογενείς πηγές και η διασύνδεση συστημάτων διαφορετικού σχεδιασμού και φιλοσοφίας καθίσταται όλο και πιο επιτακτική. Η διαλειτουργικότητα μεταξύ διαφορετικών συστημάτων επιδρά πολλαπλασιαστικά στα οφέλη από την χρήση τους, πράγμα που δικαιολογεί την τεράστια

προσπάθεια που παρατηρείται στον τομέα αυτό. Μία ακόμα αδυναμία του παραδοσιακού τρόπου αποθήκευσης και διαχείρισης δεδομένων αναδεικνύεται από την έλευση όλο και περισσότερων ευφυών εφαρμογών που ξεφεύγουν από την κλασσική θεώρηση των αποθηκευμένων δεδομένων απλώς ως καταγεγραμμένων πληροφοριών. Οι απαιτήσεις που έχουν από τα συστήματα διαχείρισης δεδομένων που τις υποστηρίζουν επιβάλλουν τη θεώρηση των δεδομένων ως γνώση που μπορεί να οδηγήσει σε νέα συμπεράσματα τα οποία δεν καταγράφονται ρητά στα διαθέσιμα δεδομένα. Η αποτελεσματική υλοποίηση αυτού του είδους των υπηρεσιών ανοίγει το δρόμο σε επαναστατικές νέες εφαρμογές που θα οδηγήσουν στην αυτοματοποίηση ακόμα περισσότερων εργασιών. Το όραμα του σημασιολογικού ιστού είναι ίσως το κυριότερο παράδειγμα των δυνατοτήτων που ανοίγονται από αυτού του είδους τις τεχνολογίες. Ο στόχος είναι οι εφαρμογές του μέλλοντος να είναι σε θέση να δράσουν αυτόνομα για λογαριασμό του χρήστη παρέχοντας σημαντικά καλύτερες υπηρεσίες. Απαραίτητη προϋπόθεση είναι να καταστεί καταληπτό από αυτές το περιεχόμενο του παγκόσμιου ιστού και εν συνεχεία η ύπαρξη ικανοποιητικών υπηρεσιών συλλογιστικής για την αποτελεσματική επεξεργασία των διαθέσιμων πληροφοριών.

Η έρευνα έχει αναδείξει στην πρωτοπορία της προσπάθειας προς αυτή την κατεύθυνση τις περιγραφικές λογικές. Οι περιγραφικές λογικές είναι μια οικογένεια γλωσσών με μεγάλες δυνατότητες στην αναπαράσταση και τη διαχείριση της γνώσης στα υπολογιστικά συστήματα. Η προσαρμοστικότητα που επιδεικνύουν σε ό,τι αφορά την εκφραστικότητα και τα υπολογιστικά χαρακτηριστικά τους, τις καθιστά ιδανικές για μια ευρεία γκάμα εφαρμογών. Ένα παράδειγμα είναι η πρόσβαση σε βάσεις δεδομένων μέσω λογικής, που είναι ίσως η αποτελεσματικότερη απάντηση στο πρόβλημα της διασύνδεσης διαφορετικών συστημάτων διαχείρισης δεδομένων. Αποδίδοντας τη σημασία κάθε τμήματος μιας βάσης παρέχεται η δυνατότητα για τη χρήση της σε υψηλότερο επίπεδο, χωρίς να είναι απαραίτητη η γνώση λεπτομερειών σχετικά με την οργάνωσή της. Με αυτό τον τρόπο βάσεις δεδομένων με διαφορετικά σχήματα ή ακόμα και βάσεις που χρησιμοποιούν ασύμβατες τεχνολογίες μπορούν να παρουσιαστούν στον χρήστη ως μια ενιαία βάση γνώσης. Επιπλέον, η σημασιολογική επισήμανση των πόρων του σημασιολογικού ιστού, μέσω φορμαλισμών που βασίζονται στις περιγραφικές λογικές, ανοίγει το δρόμο σε ευφυείς εφαρμογές.

Μέρος της προσπάθειας προς την κατεύθυνση που μόλις περιγράφηκε, αποτελεί το Rapid. Πρόκειται για ένα εργαλείο που επιτυγχάνει την ταχύτατη υλοποίηση υπηρεσιών εξαγωγής συμπερασμάτων, ιδιαίτερα σε βάσεις γνώσης που συνοδεύονται από εκτεταμένα σύνολα δεδομένων. Υποστηρίζει τις περιγραφικές λογικές DL-lite και ELHI οι οποίες διακρίνονται από καλή ισορροπία μεταξύ εκφραστικότητας και υπολογιστικού κόστους των υπηρεσιών συλλογιστικής. Το Rapid επιστρατεύει την τεχνική της μεταγραφής για την εξυπηρέτηση ερωτημάτων προς τη βάση γνώσης. Η διαδικασία απάντησης πραγματοποιείται σε δύο

στάδια. Αρχικά, λαμβάνεται υπόψη το ερώτημα και οι περιορισμοί που επιβάλλει η κωδικοποιημένη, στις προαναφερθείσες περιγραφικές λογικές, γνώση. Με αυτά τα δύο συστατικά δημιουργείται η μεταγραφή του ερωτήματος. Στο δεύτερο στάδιο γίνεται η αποτίμηση της μεταγραφής πάνω στο σύνολο δεδομένων της βάσης γνώσης για την εξαγωγή του αποτελέσματος.

Η παρούσα εργασία αφορά το δεύτερο στάδιο της διαδικασίας απάντησης ερωτημάτων μέσω μεταγραφής. Σκοπός ήταν η δημιουργία μιας εφαρμογής που θα είναι σε θέση να αποτιμήσει την μεταγραφή που παράγεται από το Rapid. Η δημιουργία μιας εφαρμογής από το μηδέν ειδικά για το Rapid επιτρέπει τον λεπτομερέστερο έλεγχο των χαρακτηριστικών της παραγόμενης μεταγραφής, πράγμα που συμβάλει καθοριστικά και στην αξιολόγηση της. Απαραίτητη προϋπόθεση είναι η ύπαρξη μεγάλης ευελιξίας όσον αφορά στα συστήματα διαχείρισης βάσεων δεδομένων τα οποία είναι δυνατό να χρησιμοποιηθούν για την αποτίμηση της μεταγραφής. Με τη δυνατότητα υποστήριξης νέων συστημάτων με την ελάχιστη δυνατή προσπάθεια δίνεται η ευκαιρία δοκιμής και χρήσης του Rapid σε πολλά διαφορετικά περιβάλλοντα. Ακόμα μια απαίτηση είναι να δίνεται η δυνατότητα αποτίμησης ερωτημάτων πάνω σε ιδιαίτερα εκτεταμένα σύνολα δεδομένων χωρίς να είναι απαραίτητη η κατοχή προηγμένου εξοπλισμού από την πλευρά του χρήστη. Σε πρώτη φάση υποστηρίζονται ο εξυπηρετητής βάσης RDF Fuseki και το σύστημα διαχείρισης σχεσιακής βάσης δεδομένων PostgreSQL. Η αποθήκευση δεδομένων ως τριάδων RDF είναι μια από τις τεχνολογίες που αποτελούν πρόταση του W3C για την πραγματοποίηση του σημασιολογικού ιστού και είναι κοντά στις απαιτήσεις που έχουν οι περιγραφικές λογικές από τις δομές αποθήκευσης δεδομένων. Από την άλλη πλευρά, οι σχεσιακές βάσεις είναι ίσως η πλέον διαδεδομένη τεχνολογία για την οποία υπάρχει πολυετής εμπειρία και υποστηρίζεται από πολλά ώριμα συστήματα.

Διαβάζοντας τα κεφάλαια που ακολουθούν ο αναγνώστης μπορεί να σχηματίσει μια πλήρη εικόνα για τις σχετικές τεχνολογίες και τα συστήματα που τις υλοποιούν. Στο επόμενο κεφάλαιο παρουσιάζονται οι τεχνολογίες πίσω από τις βάσεις γνώσης και οι προσεγγίσεις που έχουν επιχειρηθεί για την εκμετάλλευσή τους. Είναι εντυπωσιακή η ποικιλία των τεχνικών που έχουν αναδειχθεί από την έρευνα των τελευταίων δεκαετιών. Επίσης παρουσιάζονται συνοπτικά τα κυριότερα εργαλεία, παλιότερα και νέα, εμπορικά και ερευνητικού χαρακτήρα, που υπάρχουν σήμερα και σχετίζονται με την εξαγωγή συμπερασμάτων. Στο τρίτο κεφάλαιο παρουσιάζεται η γλώσσα datalog. Τη θεωρητική περιγραφή της ακολουθεί ο αλγόριθμος αποτίμησης των προγραμμάτων datalog και δύο πολύ σημαντικές τεχνικές βελτιστοποίησης που χρησιμοποιούνται από την εφαρμογή. Στο τέταρτο κεφάλαιο παρουσιάζονται τα κύρια σημεία της εφαρμογής που υλοποιήθηκε στα πλαίσια της παρούσας εργασίας. Στη συνέχεια

ακολουθεί ο σχολιασμός αποτελεσμάτων από την εκτέλεση της εφαρμογής, τα συμπεράσματα και οι προτάσεις για μελλοντικές επεκτάσεις.

# 2

# *Αναπαράσταση και διαχείριση γνώσης*

Ο μηχανισμός αναπαράστασης γνώσης στον οποίο βασίζονται οι βάσεις γνώσης με τις οποίες θα ασχοληθούμε είναι περιγραφικές λογικές[1] [1] (ΠΛ στο εξής). Οι ΠΛ αποτελούν μία οικογένεια γλωσσών αναπαράστασης γνώσης και επινοήθηκαν τις τελευταίες τρεις δεκαετίες από επιστήμονες που δραστηριοποιούνται στο πεδίο της πληροφορικής. Η κινητήριος δύναμη πίσω από την έρευνα για τις ΠΛ ήταν η ανάγκη για έναν εκφραστικό, τυπικό, εύληπτο και εύχρηστο τρόπο για την αναπαράσταση της γνώσης. Η απαίτηση για εκφραστικότητα είναι εύλογη για να είναι δυνατή η αναπαράσταση πολύπλοκων πληροφοριών. Παράλληλα, ένας φορμαλισμός αναπαράστασης γνώσης οφείλει να έχει ξεκάθαρα ορισμένη σημασιολογία. Για παράδειγμα, η φυσική γλώσσα που χρησιμοποιείται στην καθημερινή επικοινωνία είναι ακατάλληλη για την τυπική αναπαράσταση γνώσης λόγω της απουσίας αυστηρά ορισμένης σημασιολογίας. Η πρόταση "χτύπησε το παιδί με το ξύλο" μπορεί να σημαίνει είτε ότι το ξύλο χρησιμοποιήθηκε σαν όπλο είτε ότι το θύμα κρατούσε ένα ξύλο. Αυτού του είδους η αβεβαιότητα δεν είναι αποδεκτή καθώς οδηγεί σε απώλεια πληροφορίας. Επιπλέον, αν η χρησιμοποιούμενη γλώσσα είναι απλή και εύληπτη διευκολύνεται η διαδικασία δημιουργίας και συντήρησης πρακτικών εφαρμογών από ανθρώπους χωρίς ιδιαίτερη εξοικείωση με την αυστηρή μαθηματική λογική. Τέλος, επειδή ο τελικός στόχος δεν είναι μόνον η καταγραφή

[1] Αν και ο κυρίαρχος όρος είναι Περιγραφικές Λογικές (Description Logics) στη διεθνή βιβλιογραφία χρησιμοποιούνται επίσης οι όροι «terminological logics», «term subsumption systems» και «taxonomic logics».

της γνώσης, οι γλώσσες αναπαράστασης γνώσης συνοδεύονται και από μηχανισμούς εξαγωγής συμπερασμάτων. Στις εφαρμογές που μας ενδιαφέρουν η διαχείριση της καταγεγραμμένης γνώσης και η εξαγωγή συμπερασμάτων από αυτήν είναι απαραίτητο να πραγματοποιείται από μηχανές χωρίς ανθρώπινη παρέμβαση και για το λόγο αυτό μία ακόμα προϋπόθεση είναι η χαμηλή πολυπλοκότητα, η "υπολογιστική ευχρηστία".

Πέρα από τις ΠΛ υπάρχει πληθώρα γλωσσών αναπαράστασης γνώσης με τις πιο γνωστές να είναι η Προτασιακή Λογική (Propositional Logic) και η Κατηγορηματική Λογική Πρώτης Τάξης (First Order Predicate Logic) [2] που προέρχονται από τον χώρο των μαθηματικών και τα Σημασιολογικά Δίκτυα (Semantic Networks) [3] από το πεδίο της πληροφορικής. Επίσης αξιοποίηση της γνώσης για τη δημιουργία ευφυών εφαρμογών είναι δυνατή μέσω των πλαισίων (Frames) [4] και των συστημάτων κανόνων παραγωγής (Production Rules Systems) [5]. Οι παραπάνω μέθοδοι αναπαράστασης γνώσης εμφανίζουν μια σειρά από αδυναμίες. Σε γενικές γραμμές, αυτές είναι η περιορισμένη εκφραστικότητα της Προτασιακής Λογικής η οποία επιτρέπει την καταγραφή μόνο απλών ισχυρισμών, η μη αποφασισιμότητα της Κατηγορηματικής Λογικής Πρώτης Τάξης με την οποία είναι δυνατή η καταγραφή πολύ εκφραστικών ισχυρισμών αλλά είναι αδύνατη η επεξεργασία τους χωρίς την παρέμβαση της ανθρώπινης διαίσθησης και εμπειρίας. Τα Σημασιολογικά Δίκτυα αποτελούν μια ιδιαίτερα εύληπτη, οπτικοποιημένη μορφή αναπαράστασης γνώσης που όμως δεν συνοδεύεται από αυστηρά τυποποιημένη σημασιολογία με αποτέλεσμα να περιορίζεται η χρησιμότητά τους. Τα συστήματα κανόνων παραγωγής αποτελούν ένα διαδικαστικό φορμαλισμό αναπαράστασης γνώσης καθώς εστιάζουν κυρίως στις υπολογιστικές διαδικασίες που επιλέγεται να συνδεθούν με τα αντικείμενα του υπό μοντελοποίηση κόσμου παρά με τα συμπεράσματα που προκύπτουν από αυτά. Το ίδιο ισχύει και για τα πλαίσια με τη διαφορά ότι ακολουθούν αντικειμενοστραφή προσέγγιση στην οργάνωση τους. Αυτοί είναι οι κυριότεροι λόγοι που οδήγησαν στην υιοθέτηση των ΠΛ ως τις de facto γλώσσες αναπαράστασης γνώσης σε μία ευρεία γκάμα εφαρμογών. Χαρακτηριστικά παραδείγματα είναι η πρόσβαση σε δεδομένα μέσω οντολογιών ή ο σημασιολογικός ιστός όπου υπάρχει μεγάλη κινητικότητα τα τελευταία χρόνια. Παρουσιάζονται όλο και περισσότερα συστήματα συλλογιστικής με δυνατότητες χειρισμού εκτεταμένων συνόλων δεδομένων και οριστικοποιούνται πρότυπα όπως η γλώσσα περιγραφής οντολογιών OWL, θέματα τα οποία θα εξετάσουμε στη συνέχεια. Και στις δύο περιπτώσεις η πρόοδος στην έρευνα πάνω στις ΠΛ παίζει καθοριστικό ρόλο.

## *2.1 Περιγραφικές Λογικές*

Υπάρχουν ορισμένα χαρακτηριστικά που είναι κοινά σε όλες τις ΠΛ και αποτελούν τα βασικά δομικά στοιχεία κάθε βάσης γνώσης που βασίζεται σε αυτές. Πρόκειται για διακριτά σύνολα, τα περιεχόμενα των οποίων διαφέρουν ανάλογα με την γνώση προς περιγραφή. Έχουμε το σύνολο των ατομικών εννοιών **C** που αποτελείται από τις στοιχειώδεις έννοιες οι οποίες δεν περιγράφονται μέσω άλλων εννοιών, το σύνολο των ατομικών ρόλων ή σχέσεων **R** και το σύνολο ατόμων ή σταθερών **I**. Όπως θα δούμε παρακάτω, με τη χρήση κατασκευαστών εννοιών μπορούμε να συνθέσουμε σύνθετες έννοιες ή περιγραφές εννοιών. Προκειμένου να καταστεί δυνατή η επεξεργασία πολύπλοκης γνώσης η χρήση των περιγραφών είναι απαραίτητη. Τα συστήματα λογικής που βασίζονται σε αυτές δικαιωματικά παίρνουν το όνομα «Περιγραφικές Λογικές». Στη συνέχεια, τα ονόματα εννοιών θα γράφονται με κεφαλαίο το πρώτο γράμμα ενώ αυτά των ρόλων θα ξεκινούν με μικρό. Σε πιο γενικά παραδείγματα θα χρησιμοποιούμε τα γράμματα $A$ και $B$ για να αναφερθούμε σε ατομικές έννοιες, τα $C$ και $D$ για σύνθετες έννοιες ή «περιγραφές εννοιών», τα $R$ και $S$ για ρόλους και τα μικρά γράμματα $a$ και $b$ για σταθερές. Δύο ιδιαίτερες έννοιες είναι η καθολική έννοια, $\top$, που ισοδυναμεί με το σύνολο των εννοιών, ατομικών και μη και η κενή έννοια, $\bot$.

Υπάρχει πληθώρα διαθέσιμων κατασκευαστών εννοιών, ο καθένας εκ των οποίων προσφέρει μια διαφορετική δυνατότητα περιγραφής. Οι κατασκευαστές που παρέχει κάθε περιγραφική λογική καθορίζουν την εκφραστικότητά της, δηλαδή τις έννοιες που μπορούν να περιγραφούν από αυτήν. Μία από τις πιο βασικές περιγραφικές λογικές είναι η $\mathcal{AL}$. Το σύνολο των κατασκευαστών που επιτρέπονται είναι {¬, ⊓, ∀, ∃}. Οι τρόποι σύνθεσης περιγραφών εννοιών ή σύνθετων εννοιών είναι:

$$C, D \rightarrow A \mid \neg A \mid \top \mid \bot \mid C \sqcap D \mid \forall R.C \mid \exists R.\top$$

Η παραπάνω έκφραση είναι επαγωγική, στην περιγραφή μιας σύνθετης έννοιας μπορούν να μετέχουν και άλλες σύνθετες έννοιες ή ακόμα και η ίδια. Ένα παράδειγμα σύνθετης έννοιας στην $\mathcal{AL}$ είναι η $Άνθρωπος \sqcap \exists έχειΠαιδί.\top$ που περιγράφει την έννοια γονέας ως άνθρωπος για τον οποίο υπάρχει σύνδεση με την σχέση $έχειΠαιδί$ με κάποιο άτομο.

Ένα από τα κυριότερα προτερήματα των περιγραφικών λογικών είναι ότι το νόημα των εκφράσεων είναι απολύτως ξεκάθαρο αφού ορίζεται με μαθηματική αυστηρότητα. Ονόματα όπως τα $Άνθρωπος$ και $έχειΠαιδί$ που μόλις είδαμε είναι απλώς ετικέτες που βοηθούν τον χρήστη να διατηρεί την εποπτεία της βάσης γνώσης. Οποιαδήποτε άλλη συμβολοσειρά θα μπορούσε να είχε επιλεγεί στη θέση τους χωρίς να υπάρχει η παραμικρή επίπτωση στο νόημα. Η συγκεκριμενοποίηση του νοήματος των προτάσεων των ΠΛ επιτυγχάνεται μέσω

της ερμηνείας $I$ που ορίζεται ως ένα ζεύγος $(\Delta^I, \cdot^I)$ και συνοδεύει την καταγεγραμμένη γνώση. Το $\Delta^I$είναι ένα μη κενό σύνολο που ονομάζεται χώρος ερμηνείας[2] ενώ $\cdot^I$ είναι η συνάρτηση ερμηνείας[3]. Υπάρχει μεγάλη ευελιξία ως προς τα στοιχεία που αποτελούν το σύνολο $\Delta^I$ καθώς αυτά συμβολίζουν όλα τα στοιχεία του κόσμου που παρουσιάζουν ενδιαφέρον. Μπορεί να υπάρχουν στοιχεία του $\Delta^I$ που αντιστοιχούν σε πρόσωπα, πράγματα, μέρη, αφηρημένες έννοιες κλπ. Κάθε σταθερά $a$ αποτελεί ένα όνομα στο οποίο η συνάρτηση ερμηνείας αντιστοιχεί ένα συγκεκριμένο στοιχείο του $\Delta^I$. Το στοιχείο αυτό θα το συμβολίζουμε με $a^I$ και αποτελεί την ερμηνεία της σταθεράς. Για παράδειγμα, η σταθερά «Δήμητρα» μπορεί να αντιστοιχιστεί σε έναν συγκεκριμένο άνθρωπο ή σε μια βάρκα ανάλογα με την χρησιμοποιούμενη ερμηνεία. Η συνάρτηση ερμηνείας αντιστοιχεί κάθε έννοια $C$ σε ένα υποσύνολο του $C^I$ του χώρου ερμηνείας ($C^I \supseteq \Delta^I$ ), ενώ κάθε ρόλος $R$ αντιστοιχίζεται στο υποσύνολο $R^I$ του $\Delta^I \times \Delta^I$ ($R^I \supseteq \Delta^I \times \Delta^I$). Άρα, οι έννοιες αντιπροσωπεύουν κατηγορίες αντικειμένων και οι ρόλοι δυαδικές σχέσεις ανάμεσα σε αυτά.

Η συνάρτηση ερμηνείας χρησιμοποιείται και για σύνθετες έννοιες επεκτεινόμενη κατάλληλα για κάθε κατασκευαστή.

**Πίνακας 1: Ερμηνεία σύνθετων εννοιών AL.**

| Σύνθετη έννοια | Ερμηνεία |
|---|---|
| $(\neg A)^I$ | $\Delta^I \setminus A^I$ |
| $\top^I$ | $\Delta^I$ |
| $\bot^I$ | $\emptyset$ |
| $(C \sqcap D)^I$ | $C^I \cap D^I$ |
| $(\forall R.C)^I$ | $\{a \in \Delta^I \mid \forall b \in \Delta^I.(a,b) \in R^I \rightarrow b \in C^I\}$ |
| $(\exists R.\top)^I$ | $\{a \in \Delta^I \mid \exists b \in \Delta^I.(a,b) \in R^I\}$ |

Ο ¬ είναι ο κατασκευαστής άρνησης που στην $\mathcal{AL}$ χρησιμοποιείται μόνο μπροστά από ατομικές έννοιες. Το $\neg A$ συμβολίζει την άρνηση της έννοιας $A$ και η ερμηνεία του περιλαμβάνει τα στοιχεία του $\Delta^I$ που δεν περιλαμβάνει η ερμηνεία της έννοιας $A$. Ο κατασκευαστής της τομής $\sqcap$ κατασκευάζει μια νέα έννοια από την τομή δύο άλλων και ερμηνεύεται από το σύνολο των κοινών στοιχείων των δύο εννοιών στις οποίες εφαρμόζεται. Η ερμηνεία της καθολικής έννοιας περιλαμβάνει το σύνολο των στοιχείων του χώρου ερμηνείας ενώ αυτή της κενής δεν περιλαμβάνει κανένα. Και οι δύο αποτελούν συντακτικές

[2] Domain of interpretation

[3] Interpretation function

συντομεύσεις σύνθετων εννοιών. Η κενή έννοια $\perp$ αντιστοιχεί στην $C \sqcap \neg C$ η οποία είναι κενή σε όλες τις πιθανές ερμηνείες. Η $C$ μπορεί να είναι οποιαδήποτε έννοια, ατομική ή μη. Η έννοια $\forall R.C$ ονομάζεται περιορισμός τιμής και η ερμηνεία της αντιστοιχεί στο σύνολο των στοιχείων τα οποία αν συνδέονται με την σχέση $R$ με ένα δεύτερο στοιχείο τότε αυτό ανήκει υποχρεωτικά στην ερμηνεία της έννοιας $C$. Αξίζει να σημειωθεί ότι και στοιχεία τα οποία δεν συνδέονται με άλλα μέσω της σχέσης $R$, επίσης περιλαμβάνονται στην ερμηνεία της $\forall R.C$. Τέλος, έχουμε την έννοια του περιορισμένου υπαρξιακού περιορισμού, $\exists R.\top$, με την ερμηνεία της να περιλαμβάνει όσα στοιχεία συνδέονται με κάποιο άλλο μέσω της σχέσης $R$.

Μπορούμε να δημιουργήσουμε εκφραστικότερες ΠΛ προσθέτοντας επιπλέον κατασκευαστές εννοιών. Αν προσθέσουμε τον τελεστή ένωσης $\sqcup$, εφαρμόσιμο σε σύνθετες έννοιες, παίρνουμε την ΠΛ $\mathcal{ALU}$. Η ερμηνεία του τελεστή είναι $(C \sqcup D)^I = C^I \cup D^I$. Η έννοια $\top$ ορίζεται από την σύνθετη έννοια $C \sqcup \neg C$ η οποία περιλαμβάνει το σύνολο των στοιχείων του $\Delta^I$ για κάθε δυνατή ερμηνεία. Η ΠΛ $\mathcal{ALE}$ χρησιμοποιεί τον πλήρη υπαρξιακό περιορισμό $\mathcal{E}$ ο οποίος επεκτείνει τον περιορισμένο υπαρξιακό περιορισμό με τη δυνατότητα να ορίζεται η έννοια της οποίας θα είναι μέλος το δεύτερο στοιχείο που θα συμμετέχει στην σχέση. Πιο αναλυτικά η ερμηνεία του $\mathcal{E}$ ορίζεται ως εξής:

$$(\exists R.C)^I = \{a \in \Delta^I \mid \exists b \in \Delta^I.(a,b) \in R^I \rightarrow b \in C^I\}$$

Με την $\mathcal{ALE}$ μπορούμε να περιγράψουμε τους γονείς αγοριών ως εξής: $Άνθρωπος \sqcap \exists ΈχειΠαιδί.Αρσενικό$. Άλλες επεκτάσεις περιορίζουν τον αριθμό των ατόμων μέσω των οποίων μπορεί να συνδέεται ένα άτομο. Ο συναρτησιακός περιορισμός πληθικότητας, $\mathcal{F}$, δίνει σύνθετες έννοιες του τύπου $\leq 1R$ και $\geq 1R$ με ερμηνεία:

$$(\leq 1R)^I = \{a \in \Delta^I \mid \#\{b|(a,b) \in R^I\} \leq 1\}$$

$$(\geq 1R)^I = \{a \in \Delta^I \mid \#\{b|(a,b) \in R^I\} \geq 1\}$$

δηλαδή το πολύ ένα και μέχρι ένα άλλο στοιχείο μπορεί να συμμετέχει στη σχέση R με τα στοιχεία που ικανοποιούν την συγκεκριμένη σύνθετη έννοια. Φυσική επέκταση αυτού του περιορισμού είναι να επιτρέπεται στην θέση του 1 να μπει οποιοσδήποτε αριθμός δίνοντας τις έννοιες $\leq nR$ και $\geq nR$. Έτσι παίρνουμε τον περιορισμό πληθικότητας, $\mathcal{N}$, με ερμηνεία:

$$(\leq nR)^I = \{a \in \Delta^I \mid \#\{b|(a,b) \in R^I\} \leq n\}$$

$$(\geq nR)^I = \{a \in \Delta^I \mid \#\{b|(a,b) \in R^I\} \geq n\}$$

Η επόμενη λογική επέκταση του $\mathcal{N}$ είναι με την προσθήκη της δυνατότητας να ορίζεται μια συγκεκριμένη έννοια στην οποία θα πρέπει να μετέχει το δεύτερο στοιχείο. Έτσι

καταλήγουμε στον προσοντούχο περιορισμό πληθικότητας[4] που συμβολίζεται με $\mathcal{Q}$. Η ερμηνεία σε απόλυτη αντιστοιχία με τα προηγούμενα είναι:

$$(\leq nR.C)^I = \{a \in \Delta^I \mid \#\{b|(a,b) \in R^I \ \& \ b \in C\} \leq n\}$$

$$(\geq nR.C)^I = \{a \in \Delta^I \mid \#\{b|(a,b) \in R^I \ \& \ b \in C\} \geq n\}$$

Ενίσχυση της εκφραστικότητας έχουμε αν επιτρέψουμε άρνηση σε σύνθετες έννοιες. Ο κατασκευαστής αυτός συμβολίζεται με $C$ και η ερμηνεία του είναι:

$$(\neg C)^I = \Delta^I \setminus C^I$$

Μια διαφορετικού τύπου επέκταση επιτυγχάνεται επιτρέποντας τη δημιουργία εννοιών από σταθερές. Αυτού του είδους οι έννοιες αποκαλούνται ονοματικές έννοιες[5] και η υποστήριξή τους υποδηλώνεται από το $O$ στο όνομα της ΠΛ. Από τις σταθερές $a$ και b δημιουργούνται οι έννοιες $\{a\}$ και $\{b\}$. Η ερμηνεία ορίζεται ως το υποσύνολο του $\Delta^I$ που περιέχει μόνο το στοιχείο $a^I$ που αποτελεί την ερμηνεία του $a$, δηλαδή $\{a\}^{\mathrm{I}} = \{a^I\}$. Για παράδειγμα με τη χρήση της ονοματικής έννοιας Μ.Αλέξανδρος μπορούμε να περιγράψουμε όλες τις πόλεις που έχουν κατακτηθεί από αυτόν με την έκφραση Πόλη ⊓ ∃έχειΚατακηθεί.Μ.Αλέξανδρος.

Η συνεχής προσθήκη κατασκευαστών δεν αυξάνει την εκφραστικότητα σε όλες τις περιπτώσεις καθώς είναι δυνατό η ίδια έννοια να περιγραφεί με διαφορετικό τρόπο. Σύμφωνα με τους κανόνες De Morgan της θεωρίας συνόλων ισχύει η ισότητα $\neg(\neg C \sqcap \neg D) \equiv C \sqcup D$ που σημαίνει ότι ο κατασκευαστής της άρνησης, σε συνδυασμό με αυτόν της τομής, μπορεί να δώσει ισοδύναμες εκφράσεις με αυτές του κατασκευαστή της ένωσης $\mathcal{U}$. Επιπλέον, ισχύει και η ισότητα $\neg(\forall R.C) \equiv \neg\forall R.\neg C \equiv \exists R.C$ άρα ο κατασκευαστής $C$, όταν είναι διαθέσιμοι και άλλοι κατασκευαστές της $\mathcal{AL}$, επαρκεί για να επιτευχθεί η εκφραστικότητα του πλήρους υπαρξιακού περιορισμού. Συνεπώς, ΠΛ $\mathcal{ALC}$ παρέχει την ίδια εκφραστικότητα με την $\mathcal{ALUE}C$. Ένας άλλος ισχυρός κατασκευαστής είναι ο $\mathcal{Q}$, ο οποίος επίσης είναι σε θέση να εκφράσει τον πλήρη υπαρξιακό περιορισμό $\mathcal{E}$ αφού ισχύει $\geq 1R.C \equiv \exists R.C$. Το ίδιο συμβαίνει και με τον περιορισμό τιμής λόγω της $\forall R.C \equiv \leq 0R.\neg C$.

Οι ΠΛ μας επιτρέπουν να δίνουμε ονόματα στις σύνθετες έννοιες που δημιουργούμε με τους κατασκευαστές. Αυτό είναι δυνατό μέσω των αξιωμάτων ισοδυναμίας, $C \equiv D$. Για να ικανοποιούνται τα αξιώματα αυτού του τύπου, πρέπει να ισχύει $C^I = D^I$, δηλαδή η ερμηνεία της έννοιας $C$ οφείλει να ταυτίζεται με αυτήν της $D$. Επιπροσθέτως, μπορούμε να ορίσουμε

[4] Στη διεθνή βιβλιογραφία αναφέρεται ως qualified number restriction ή qualified cardinality restriction.

[5] Nominal concepts

σχέσεις υπαγωγής ανάμεσα στις έννοιες χρησιμοποιώντας αξιώματα υπαγωγής[6], $C \sqsubseteq D$. Τα αξιώματα υπαγωγής απαιτούν από τις ερμηνείες που τα ικανοποιούν να ερμηνεύουν την έννοια στα αριστερά ως υποσύνολο της ερμηνείας που δίνουν στην έννοια στο δεξί μέρος, δηλαδή $C^I \subseteq D^I$. Το αξίωμα ισοδυναμίας $C \equiv D$ είναι ισοδύναμο με τα αξιώματα $C \sqsubseteq D$ και $D \sqsubseteq C$. Αν στη βάση γνώσης υπάρχει ένα αξίωμα $C \equiv D$ τότε η έννοια $C$ θεωρείται πλήρως ορισμένη καθώς έχουμε το σύνολο των αναγκαίων και ικανών συνθηκών που πρέπει να πληρούνται από μια οντότητα για να ανήκει στην έννοια $C$. Αν έχουμε μόνο το αξίωμα υπαγωγής $C \sqsubseteq D$ τότε η $C$ θεωρείται πρωταρχική έννοια διότι έχουμε μόνο ένα σύνολο αναγκαίων συνθηκών. Με την χρήση αξιωμάτων υπαγωγής μπορούμε να δημιουργήσουμε ιεραρχίες εννοιών που είναι απαραίτητες για την καταγραφή γνώσης. Τα αξιώματα που περιλαμβάνουν κάποια σύνθετη έννοια στο αριστερό μέρος χαρακτηρίζονται ως υπαγωγές γενικών εννοιών ή γενικευμένα αξιώματα[7].

Πέρα από τη χρήση όλο και πιο σύνθετων εννοιών, η εκφραστικότητα μπορεί να ενισχυθεί και με επεκτάσεις που αφορούν τους ρόλους. Σε πλήρη αντιστοιχία με τις έννοιες, παρέχεται η δυνατότητα δημιουργίας ιεραρχίας ρόλων, η οποία συμβολίζεται με το γράμμα $\mathcal{H}$. Τα αξιώματα υπαγωγής ρόλων δηλώνονται και ικανοποιούνται με αντίστοιχο με τις έννοιες τρόπο. Η έκφραση $R \sqsubseteq S$ απαιτεί ο ρόλος $R$ να είναι υπορόλος του $S$ με τις ερμηνείες που την ικανοποιούν να ερμηνεύουν τον ρόλο $R$ με ένα υποσύνολο του συνόλου $S^I$ που αποτελεί την ερμηνεία του ρόλου $S$, δηλαδή να ισχύει $R^I \subseteq S^I$. Η ιεραρχία ρόλων μπορεί να επεκταθεί επιτρέποντας αξιώματα υπαγωγής σύνθετων ρόλων όπως το έχειΓονέα $\circ$ έχειΑδελφό $\sqsubseteq$ έχειΘείο που σημαίνει ότι δύο αντικείμενα που συνδέονται μέσω της αλυσίδας ρόλων στα αριστερά πρέπει να συνδέονται και απευθείας μέσω του ρόλου στα δεξιά. Σε αυτή την περίπτωση, στο όνομα της ΠΛ περιλαμβάνει το $\mathcal{R}$**.** Επιπλέον, υπάρχει η δυνατότητα ορισμού μεταβατικών ρόλων. Οι γλώσσες που επιτρέπουν αξιώματα μεταβατικών ρόλων διακρίνονται από το σύμβολο $R^+$ που ενσωματώνεται σαν δείκτης στο όνομά τους. Για παράδειγμα, αν στη γλώσσα $\mathcal{ALC}$ προστεθεί υποστήριξη μεταβατικών ρόλων [6] προκύπτει η $\mathcal{ALC}_{R^+}$, η οποία συμβολίζεται με το γράμμα $\mathcal{S}$[8] για την αποφυγή μακρόσυρτων ονομάτων. Ένας μεταβατικός ρόλος $R$ ερμηνεύεται με τον παρακάτω τρόπο: αν $\{(a^I, b^I), (b^I, c^I)\} \subseteq R^I$ τότε $(a^I, c^I) \in R^I$. Ένας ρόλος όπως ο έχωΠρόγονο θα δηλωθεί ως μεταβατικός με την έκφραση: **Tr**(έχωΠρόγονο). Μία άλλη ιδιότητα ρόλων είναι οι αντίστροφοι ρόλοι, η ύπαρξη των

[6] Subsumption ή inclusion axioms

[7] General Concept Inclusion axioms - GCI

[8] Η επιλογή του γράμματος S λόγω της σχέσης της συγκεκριμένης ΠΛ με την προτασιακή πολυτροπική λογική S4(m) [112]

οποίων επιδεικνύεται με το γράμμα $\mathcal{I}$ ενώ η έκφραση που δίνει τον αντίστροφο ενός ρόλου $R$ είναι $R^-$. Η σημασιολογία των αντίστροφων ρόλων είναι: $(a,b) \in R^I$ ανν $(b,a) \in (R^-)^I$, δηλαδή αν το $a$ συνδέεται με το $b$ μέσω του ρόλου $R$ τότε το $b$ συνδέεται με το $a$ μέσω του ρόλου $R^-$. Μπορούμε να ορίσουμε έναν ρόλο ως τον αντίστροφο ενός άλλου, αν επιτρέπονται αξιώματα υπαγωγής ρόλων, με το αξίωμα: $\text{έχωΑπόγονο} \sqsubseteq \text{έχωΠρόγονο}^-$. Το αξίωμα $R \sqsubseteq S$ συνεπάγεται το $R^- \sqsubseteq S^-$ όταν υποστηρίζεται η ιδιότητα των αντίστροφων ρόλων. Άλλοι περιορισμοί ρόλων μπορεί να είναι η συμμετρία/ασυμμετρία, η ανακλαστικότητα ή μη και η επισήμανση δύο ρόλων ως ξένων. Ένας συμμετρικός ρόλος μπορεί να δηλωθεί ως **Sym**$(R)$ και έχει την ερμηνεία $R^I = (R^-)^I$. Η ασυμμετρία δηλώνεται με **Asym**$(R)$ και ερμηνεύεται με την $R^I \cap (R^-)^I = \emptyset$. Το γεγονός ότι ο ρόλος $R$ είναι ανακλαστικός επισημαίνεται με **Ref**$(R)$ και ερμηνεύεται ως $(x,x) \in R^I, \forall x \in \Delta^I$ ενώ η μη-ανακλαστικότητα γράφεται **Irref**$(S)$ με ερμηνεία $(x,x) \notin S^I, \forall x \in \Delta^I$. Δύο ρόλοι μπορούν να επισημανθούν ως ξένοι με την έκφραση **Dis**$(R,S)$ που ισχύει όταν $R^I \cap S^I = \emptyset$.

Οι περιγραφές και οι ιεραρχίες εννοιών και ρόλων δεν επαρκούν για την πλήρη αναπαράσταση της γνώσης. Μια βάση γνώσης πρέπει να είναι σε θέση να αναπαραστήσει πληροφορία που σχετίζεται με τα αντικείμενα του κόσμου που μοντελοποιείται κάθε φορά. Η ανάγκη αυτή καλύπτεται από τους ισχυρισμούς. Όπως είναι αναμενόμενο, απαιτούνται δύο είδη ισχυρισμών, ένα για έννοιες και ένα για ρόλους. Οι ισχυρισμοί εννοιών γράφονται $C(a)$[9] και υποδηλώνουν ότι το αντικείμενο το οποίο αποτελεί την ερμηνεία της σταθεράς $a$ αποτελεί στιγμιότυπο της έννοιας $C$. Δηλαδή για να ικανοποιείται ο ισχυρισμός $C(a)$ πρέπει το $a^I$ να περιλαμβάνεται στην ερμηνεία της έννοιας $C$, δηλαδή $a^I \in C^I$. Οι ισχυρισμοί ρόλων γράφονται $R(a,b)$[10] και ικανοποιούνται όταν τα $a^I, b^I$ που αποτελούν τις ερμηνείες των σταθερών $a, b$ συνδέονται μέσω του ρόλου $R$, δηλαδή $(a^I, b^I) \in R^I$.

Οι ΠΛ παρέχουν επιπλέον την δυνατότητα ορισμού τύπων δεδομένων. Υπάρχουν διάφορες μέθοδοι που επιτρέπουν την ενσωμάτωση τέτοιων δυνατοτήτων. Μια ευρέως χρησιμοποιούμενη είναι η type system που επισημαίνεται στο όνομα της ΠΛ με το γράμμα **D**. Ορίζεται από το ζεύγος ($\Delta_\mathbf{D}$, $\Phi_\mathbf{D}$) με το $\Phi_\mathbf{D}$ να αποτελεί το σύνολο των ονομάτων των διαθέσιμων τύπων δεδομένων και το $\Delta_\mathbf{D}$ το χώρο ερμηνείας τους. Σε πλήρη αντιστοιχία με όσα ισχύουν για το $\Delta^I$, κάθε όνομα τύπου δεδομένων $p \in \Phi_\mathbf{D}$ ερμηνεύεται με αντιστοίχηση σε ένα σύνολο $p^D$ του $\Phi_\mathbf{D}$. Σε αντίθεση με τα όσα ισχύουν για σταθερές, έννοιες και ρόλους τα ονόματα των τύπων δεδομένων έχουν μόνο μία ερμηνεία η οποία παραμένει σταθερή. Για παράδειγμα, ο τύπος integer αναφέρεται πάντα στο υποσύνολο του $\Delta_\mathbf{D}$ που περιλαμβάνει τους

[9] Στη βιβλιογραφία χρησιμοποιείται και η εναλλακτική σύνταξη $a: C$

[10] Η εναλλακτική σύνταξη για ισχυρισμούς ρόλων είναι $(a,b): R$

ακέραιους αριθμούς. Πρέπει να επισημανθεί ότι τα $\Delta_{\mathbf{D}}$ και $\Delta^I$ είναι ξένα σύνολα. Οι τύποι δεδομένων χρησιμοποιούνται μέσω των εκφράσεων του περιορισμού τιμής και του υπαρξιακού περιορισμού στις οποίες μετέχουν ρόλοι που αποκαλούνται «ενισχυμένοι» (concrete). Έστω ένας concrete role $T$ και ένα όνομα τύπου δεδομένων $p$, οι εκφράσεις $\exists T.p$ και $\forall T.p$ αποτελούν έννοιες και οι ερμηνείες τους δίνονται από τις σχέσεις:

$$(\exists T.p)^I = \{a \in \Delta^I \mid \exists b.(a,b) \in T^I \text{ και } b \in p^D\}$$

$$(\forall T.p)^I = \{a \in \Delta^I \mid \forall b \in \Delta_{\mathbf{D}}.(a,b) \in T^I \rightarrow b \in p^D\}$$

Ένα παράδειγμα αυτής της επέκτασης είναι ο concrete role «έχειΔιεύθυνση». Το $p^D$ σε αυτή την περίπτωση είναι το σύνολο των συμβολοσειρών του $\Delta_{\mathbf{D}}$ και μία έννοια θα μπορούσε να είναι η $\exists$έχειΔιεύθυνση.$"Τροίας\ 4"$.

Τέλος πρέπει να αναφερθεί ότι έχει αναζητηθεί πληθώρα επεκτάσεων για τις ΠΛ που ξεφεύγουν από τα πλαίσια του παρόντος κειμένου. Παραδείγματα επεκτάσεων πέρα από τα όρια της κλασσικής λογικής πρώτης τάξης αποτελούν οι ΠΛ με χρονικούς κατασκευαστές [7], οι ασαφείς (fuzzy), οι πιθανοκρατικές (probabilistic) και οι ΠΛ με τροπικούς (modal) κατασκευαστές [8].

## *2.2 Βάσεις γνώσης*

Η πληροφορία στις βάσεις γνώσης οργανώνεται σε διακριτά μέρη. Αξιώματα που αναφέρονται σε έννοιες, όπως αυτά της υπαγωγής και της ισοδυναμίας που παρουσιάστηκαν νωρίτερα, αποτελούν το σώμα ορολογίας (Terminological Box, TBox) που συμβολίζεται με το γράμμα $\mathcal{T}$. Κάθε ερμηνεία $\mathcal{I}$ που ικανοποιεί όλα τα αξιώματα που περιλαμβάνονται στο TBox $\mathcal{T}$ λέμε ότι το ικανοποιεί και αποτελεί ένα μοντέλο του $\mathcal{T}$. Μπορούμε να χαρακτηρίσουμε τα σώματα ορολογίας ανάλογα με τον τρόπο που χρησιμοποιούνται οι έννοιες κατά τη δημιουργία αξιωμάτων. Έχουμε τα απλά στα οποία επιτρέπονται αξιώματα της μορφής $A \equiv C$ και $A \sqsubseteq C$, όπου $A$ είναι μια ατομική έννοια ενώ στη θέση του $C$ μπορεί να βρίσκεται και σύνθετη. Αυτά που περιέχουν κύκλους αποκαλούνται κυκλικά. Ένα TBox περιέχει κύκλο εάν αντικαθιστώντας τις έννοιες που βρίσκονται στο δεξί μέρος ενός αξιώματος χρησιμοποιώντας άλλα αξιώματα του TBox είναι δυνατό να καταλήξουμε με ένα αξίωμα που περιλαμβάνει την ίδια έννοια και στις δύο πλευρές. Για παράδειγμα τα $\mathcal{T} = \{A \sqsubseteq C, C \sqsubseteq B, B \sqsubseteq A\}$ και $\mathcal{T} = \{A \sqsubseteq \exists R.A\}$ είναι κυκλικά. Μια ακόμα διάκριση μπορεί

να γίνει με βάση το εάν επιτρέπονται σύνθετες έννοιες στο αριστερό μέρος των αξιωμάτων. Κάθε TBox που περιλαμβάνει γενικευμένα αξιώματα χαρακτηρίζεται γενικευμένο.

Όμοια, τα αξιώματα που αφορούν ρόλους οργανώνονται στο σώμα ρόλων (Rule Box, RBox) το οποίο συμβολίζεται με $\mathcal{R}$. Η συνθήκη ικανοποίησης του RBox ορίζεται αντίστοιχα με αυτή του ABox. Επειδή τα αξιώματα ρόλων απαιτούν παρόμοια αντιμετώπιση με τα αξιώματα εννοιών, συνηθίζεται στην βιβλιογραφία να μην διαχωρίζεται το σώμα ρόλων από αυτό των ισχυρισμών. Η πρακτική αυτή θα ακολουθηθεί και στο παρόν κείμενο.

Οι ισχυρισμοί που διατυπώνονται για τα αντικείμενα που περιλαμβάνονται στην διαθέσιμη γνώση αποτελούν το σώμα ισχυρισμών (Assertional Box, ABox) που συμβολίζεται με $\mathcal{A}$. Πέρα από τον αναμενόμενο ορισμό της ικανοποίησης του ABox από μια ερμηνεία - μοντέλο αυτού όταν ικανοποιούνται όλοι οι ισχυρισμοί που περιλαμβάνει, υπάρχει και ένας ακόμα ορισμός. Μια ερμηνεία $\mathcal{I}$ ικανοποιεί το ABox $\mathcal{A}$ με βάση το TBox $\mathcal{T}$ αν η $\mathcal{I}$ είναι ταυτόχρονα μοντέλο και των δύο. Η δυνατότητα εξέτασης ενός ABox υπό το πρίσμα των περιορισμών κάποιου TBox είναι καίριας σημασίας. Οι ισχυρισμοί του ABox μπορούν να θεωρηθούν ως τα δεδομένα μιας κλασσικής βάσης δεδομένων με το TBox να αντιστοιχεί στην λογική διάταξη (logical schema) αυτής. Συνεπώς ο τελευταίος ορισμός μας επιτρέπει να δούμε τα ABox και TBox ως μια ενιαία βάση δεδομένων.

Μια βάση γνώσης K μπορεί να οριστεί από το ζεύγος $\mathcal{T}$ και $\mathcal{A}$, με την έκφραση K = {$\mathcal{T}$, $\mathcal{A}$}. Η K είναι ικανοποιήσιμη ή συνεπής αν υπάρχει κάποια ερμηνεία $\mathcal{I}$ που να ικανοποιεί τα $\mathcal{T}$ και $\mathcal{A}$ στο σύνολό τους. Σε αυτή την περίπτωση η $\mathcal{I}$ αποτελεί μοντέλο της K.

Ένα πρόβλημα που υπάρχει σε κάθε βάση δεδομένων είναι πως αντιμετωπίζονται ερωτήματα που αφορούν πληροφορίες που δεν καταγράφονται σε αυτήν. Ενώ δεν έχουμε λόγο να αμφισβητήσουμε την ισχύ ενός γεγονότος που περιέχεται στην βάση δεν είναι το ίδιο σαφής ο τρόπος αντιμετώπισης των γεγονότων που δεν καταγράφονται ρητά σε αυτήν. Ένας τρόπος ερμηνείας της απουσίας πληροφορίας είναι μέσω της υπόθεσης κλειστού κόσμου (closed world assumption) [9]. Σύμφωνα με αυτήν οι πληροφορίες που δεν βρίσκονται στην βάση δεδομένων θεωρείται ότι δεν ισχύουν. Συνήθως, οι κλασσικές βάσεις δεδομένων ασπάζονται την υπόθεση κλειστού κόσμου και θεωρούν την περιεχόμενη πληροφορία πλήρη με συνέπεια να αντιμετωπίζουν τις πληροφορίες αυτές ως μη αληθείς. Οι βάσεις γνώσης που βασίζονται σε ΠΛ ακολουθούν την υπόθεση ανοιχτού κόσμου (open world assumption). Αυτό σημαίνει ότι οι πληροφορίες που απουσιάζουν από μια βάση γνώσης θεωρούνται άγνωστες, δηλαδή μπορεί να είναι αληθείς ή όχι. Έτσι, το μόνο που απαιτείται από μία ερμηνεία για να αποτελεί μοντέλο της βάσης είναι να ικανοποιεί τα $\mathcal{T}$ και $\mathcal{A}$ ταυτόχρονα. Κάθε ερμηνεία μπορεί να περιέχει επιπλέον πληροφορίες για άτομα, έννοιες και ρόλους με την προϋπόθεση να μην έρχονται σε σύγκρουση με τα αξιώματα και τους ισχυρισμούς της βάσης. Αυτό έχει ως

συνέπεια να γίνονται δεκτές ως διαφορετικά μοντέλα μιας βάσης γνώσης δύο ερμηνείες που δίνουν αντίθετη αληθοτιμή στον ίδιο ισχυρισμό που δεν περιλαμβάνεται στη βάση. Καταλήγουμε επομένως με πολλαπλά και πολλές φορές αντικρουόμενα μοντέλα για μια βάση σε ΠΛ ενώ μια παραδοσιακή βάση δεδομένων έχει ως ένα και μοναδικό μοντέλο τις πληροφορίες που καταγράφονται σε αυτήν.

## *2.3 Υπηρεσίες εξαγωγής συμπερασμάτων*

Η αναπαράσταση και η αποθήκευση πληροφορίας δεν επαρκούν για την δημιουργία ευφυών συστημάτων. Απαραίτητη προϋπόθεση είναι η ικανότητα επεξεργασίας της διαθέσιμης πληροφορίας και αξιοποίησής της, χωρίς την οποία οι ΠΛ θα όριζαν απλώς ένα νέο τρόπο δημιουργίας απλών βάσεων δεδομένων. Σε αντίθεση με τις παραδοσιακές βάσεις, είναι δυνατό να καταγραφεί ένα μέρος των προτάσεων που θεωρούνται ως αληθείς, οι οποίες μέσω της συλλογιστικής, μπορούν να αποδώσουν το σύνολο των αληθών προτάσεων. Για αυτό ένα από τα κύρια χαρακτηριστικά των βάσεων γνώσης είναι οι υπηρεσίες εξαγωγής συμπερασμάτων που παρέχουν. Υπάρχουν διαφορετικού είδους υπηρεσίες για τα σώματα ορολογίας και ισχυρισμών αλλά όλες όσες θα δούμε αμέσως μετά είναι δυνατό να αναχθούν σε μία.

Οι υπηρεσίες εξαγωγής συμπερασμάτων πάνω στο σώμα ορολογίας $\mathcal{T}$ είναι οι εξής[11]:

- Ικανοποιησιμότητα (satisfiability): Μια έννοια $C$ είναι ικανοποιήσιμη αν είναι δυνατή η κατασκευή έστω και ενός μοντέλου $\mathcal{I}$ του $\mathcal{T}$ στο οποίο η ερμηνεία της δεν είναι κενή, δηλαδή αληθεύει η $C^{\mathcal{I}} \neq \emptyset$. Συνήθως, μια μη ικανοποιήσιμη έννοια που περιλαμβάνεται στο $\mathcal{T}$, εξαιρουμένης της $\bot$, αποτελεί ισχυρή ένδειξη λάθους στον σχεδιασμό της βάσης γνώσης.
- Ισοδυναμία (equivalence): Ισοδύναμες είναι οι έννοιες που έχουν ταυτόσημες ερμηνείες. Πιο αναλυτικά, οι έννοιες $C$ και $D$ είναι ισοδύναμες με βάση το $\mathcal{T}$ αν, για κάθε μοντέλο $\mathcal{I}$ του $\mathcal{T}$, ισχύει $C^{\mathcal{I}} = D^{\mathcal{I}}$. Τότε γράφουμε $\mathcal{T} \vDash C \equiv D$.
- Υπαγωγή (subsumption): Μία έννοια $C$ λέμε ότι υπάγεται στην $D$ με βάση το $\mathcal{T}$ αν η ερμηνεία της αποτελεί υποσύνολο της ερμηνείας της $D$ σύμφωνα με τα αξιώματα που ορίζονται στο $\mathcal{T}$. Όπως και με την ισοδυναμία εννοιών η σχέση

[11] Τα αυτά ισχύουν και για τους ρόλους.

$C^I \subseteq D^I$ πρέπει να ισχύει για όλα τα μοντέλα του $\mathcal{T}$. Το γεγονός ότι από το $\mathcal{T}$ προκύπτει η υπαγωγή της $C$ στην $D$ δηλώνεται από την σχέση $\mathcal{T} \vDash C \sqsubseteq D$. Ο καθολικός υπολογισμός της μερικής διάταξης των εννοιών του $\mathcal{T}$ με βάση τη σχέση υπαγωγής ονομάζεται ταξινομία.

- Ξένες έννοιες (concept disjointness): Δύο έννοιες $C$ και $D$ είναι ξένες με βάση το $\mathcal{T}$ αν οι ερμηνείες τους σε κάθε δυνατό μοντέλο $\mathcal{I}$ αυτού δεν περιλαμβάνουν κανένα κοινό αντικείμενο, δηλαδή ισχύει $C^I \cap D^I = \emptyset$.

Όλες οι ανωτέρω υπηρεσίες μπορούν να αναχθούν σε αυτή της ικανοποιησιμότητας. Η ειδοποιός διαφορά της με τις υπόλοιπες έγκειται στο ότι για την απόδειξή της αρκεί η εύρεση ενός μόνο κατάλληλου μοντέλου ενώ για τις άλλες απαιτείται ο έλεγχος όλων των μοντέλων του $\mathcal{T}$. Με τη χρήση αυτής της αναγωγής έχουμε στη διάθεσή μας έναν απλό τρόπο απόδειξης των συμπερασμάτων από όλες τις υπηρεσίες. Η αναγωγή στην υπηρεσία της ικανοποιησιμότητας επιτυγχάνεται με την εισαγωγή κατάλληλων νέων εννοιών στο $\mathcal{T}$ και εν συνεχεία τον έλεγχο της ικανοποιησιμότητάς του. Το ερώτημα αν δύο έννοιες $C$ και $D$ είναι ξένες μπορεί να αναχθεί στο ερώτημα για την ικανοποίηση της έννοιας $C \sqcap D$ καθώς τα αντικείμενα που υπάγονται σε αυτήν περιλαμβάνονται ταυτόχρονα στις ερμηνείες των δύο πρώτων. Ο έλεγχος για την υπαγωγή της έννοιας $C$ στην $D$ ανάγεται στο πρόβλημα της ικανοποιησιμότητας της έννοιας $C \sqcap \neg D$. Η ύπαρξη ενός στοιχείου του χώρου ερμηνείας που ανήκει στην ερμηνεία της έννοιας $C$ και βρίσκεται εκτός του συνόλου που αποτελεί την ερμηνεία της $D$, οπότε και ικανοποιείται η έννοια $C \sqcap \neg D$, σημαίνει ότι δεν ισχύει η υπαγωγή. Αντιθέτως, η υπαγωγή επιβεβαιώνεται αν αποδειχθεί η μη-ικανοποιησιμότητα της $C \sqcap \neg D$. Η ισοδυναμία $C \equiv D$, όπως είδαμε, ορίζεται ως η διπλή υπαγωγή $C \sqsubseteq D$ και $D \sqsubseteq C$ και εξακριβώνεται με τον παράλληλο έλεγχο της ικανοποιησιμότητας των δύο συμμετρικών εννοιών $C \sqcap \neg D$ και $D \sqcap \neg C$. Όπως βλέπουμε, για να είναι δυνατές οι αναγωγές που μόλις παρουσιάστηκαν, απαραίτητη προϋπόθεση είναι η ΠΛ να υποστηρίζει άρνηση.

Το σώμα ισχυρισμών αξιοποιείται από τις παρακάτω υπηρεσίες εξαγωγής συμπερασμάτων:

- Συνέπεια (consistency): ένα ABox $\mathcal{A}$ είναι συνεπές με βάση ένα TBox $\mathcal{T}$ εάν υπάρχει ερμηνεία $\mathcal{I}$ που να αποτελεί μοντέλο ταυτόχρονα του $\mathcal{T}$ και του $\mathcal{A}$.
- Συνεπαγωγή (entailment): ένα ABox $\mathcal{A}$ συνεπάγεται έναν ισχυρισμό $\psi$ με βάση ένα TBox $\mathcal{T}$ εάν σε κάθε δυνατό μοντέλο των $\mathcal{A}$ και $\mathcal{T}$ ο ισχυρισμός $\psi$ είναι αληθής. Η συνεπαγωγή του $\psi$ από τα $\mathcal{A}$ και $\mathcal{T}$ συμβολίζεται με $\mathcal{T}, \mathcal{A} \vDash \psi$.

Το ερώτημα της συνεπαγωγής μιας έννοιας από ένα ABox μπορεί να αναχθεί σε αυτό της συνέπειας του ABox με βάση το $\mathcal{T}$. Η συνεπαγωγή $\mathcal{T},\mathcal{A} \models \psi$ ισχύει αν και μόνο αν το $\mathcal{A} \cup \{\neg\psi\}$ είναι ασυνεπές με βάση το $\mathcal{T}$. Επιπλέον, το πρόβλημα της ικανοποιησιμότητας μιας έννοιας $C$ του TBox $\mathcal{T}$ αντιστοιχεί στο πρόβλημα της συνέπειας του ABox $\mathcal{A} = \{C(a)\}$ με βάση το $\mathcal{T}$.

Σε όλες τις παραπάνω περιπτώσεις τα σώματα ορολογίας και ισχυρισμών μπορεί να είναι κενά. Το κενό σώμα ορολογίας δεν επιβάλλει κανέναν επιπλέον περιορισμό και έτσι μία έννοια όπως η $Άνδρας \sqcap Γυναίκα$ μπορεί κάλλιστα να αληθεύει καθώς μπορεί να κατασκευαστεί μια ερμηνεία όπου ένα αντικείμενο μετέχει ταυτόχρονα και στα δύο σύνολα που ερμηνεύουν τις έννοιες $Άνδρας$ και $Γυναίκα$. Ένα σώμα ισχυρισμών όπως το $\{Άνδρας(a), Γυναίκα(a)\}$ είναι ικανοποιήσιμο με βάση το κενό TBox. Αν όμως το κενό TBox αντικατασταθεί με κάποιο που κομίζει την γνώση ότι οι έννοιες $Άνδρας$ και $Γυναίκα$ είναι ξένες, δηλαδή $Άνδρας(a) \sqcap Γυναίκα(a) \sqsubseteq \bot$, τότε τα μοντέλα που επιτρέπονται από τον περιορισμό που επιβάλλει πλέον το TBox δεν είναι δυνατό να ικανοποιήσουν την έννοια $Άνδρας \sqcap Γυναίκα$ ούτε μπορούν να αποτελέσουν μοντέλα του παραπάνω ABox, το οποίο καθίσταται πλέον ασυνεπές. Υπάρχουν επίσης οι λογικά έγκυρες προτάσεις που συνεπάγονται από το κενό TBox και ισχύουν πάντα. Μια τέτοια πρόταση είναι η $Αυτοκίνητο \sqcap Μαύρο \sqsubseteq Αυτοκίνητο$. Δεν είναι δυνατό κάποιο σώμα ισχυρισμών να ανατρέψει το γεγονός ότι η τομή δύο συνόλων είναι πάντα υποσύνολο του ενός εκ των δύο.

## *2.4 Το πρόβλημα της απάντησης ερωτημάτων*

Η εκμετάλλευση της σημασιολογικής ταξινόμησης δεδομένων μέσω μιας βάσης γνώσης K προϋποθέτει την δυνατότητα εξόρυξης πληροφορίας μέσω ερωτημάτων προς τη βάση. Η απάντηση ερωτημάτων πάνω στην K αποτελεί ένα ακόμα πρόβλημα συλλογιστικής αφού, σε αντίθεση με τις κλασσικές βάσεις δεδομένων, είναι πολύ πιθανό να απαιτηθεί εκμαίευση υπονοούμενης πληροφορίας. Η κατηγορία ερωτημάτων με την οποία θα ασχοληθούμε στο παρόν κείμενο είναι τα συζευκτικά ερωτήματα (conjunctive queries, CQ). Ένα τέτοιο ερώτημα είναι μια έκφραση $\varphi$ λογικής πρώτης τάξης που στην περίπτωσή μας κατασκευάζεται από άτομα ενός ή δύο όρων ανάλογα με το αν αυτά αναφέρονται σε έννοια ή ρόλο. Οι όροι, $t_i$, που εμφανίζονται στα άτομα είναι είτε μεταβλητές, $y_i$, είτε σταθερές, $a_i$, που αντιστοιχούν μέσω της συνάρτησης ερμηνείας σε αντικείμενα του χώρου ερμηνείας.

Πολυπλοκότερες εκφράσεις συντίθενται μέσω της λογικής σύζευξης και του υπαρξιακού περιορισμού όπως βλέπουμε παρακάτω:

$$t \quad \rightarrow \quad y_i \mid a_i$$
$$\varphi \quad \rightarrow \quad C_j(t) \mid R_j(t_{ja}, t_{jb}) \mid \varphi_1 \wedge \varphi_2 \mid \exists y_i \varphi$$

Οι μεταβλητές της $\varphi$ των οποίων οι τιμές καταγράφονται στις απαντήσεις, ονομάζονται μεταβλητές απάντησης. Ένα ερώτημα με μεταβλητές απάντησης $y_1, \dots, y_n$ συμβολίζεται με $q(y_1, \dots, y_n)$. Για μια δεδομένη ερμηνεία $\mathcal{I}$ η ανάθεση $\boldsymbol{\alpha}$ είναι μια συνάρτηση που αντιστοιχεί κάθε ελεύθερη μεταβλητή $y$ του ερωτήματος σε ένα αντικείμενο $\boldsymbol{\alpha}(y)$ του χώρου ερμηνείας $\Delta^I$ και γράφουμε $y^{I,\boldsymbol{\alpha}} = \boldsymbol{\alpha}(y)$. Για τα ονόματα ατόμων ισχύει $\alpha^{I,\boldsymbol{\alpha}} = \alpha^I$ καθώς η $\boldsymbol{\alpha}$ δεν επιδρά σε σταθερές. Το αποτέλεσμα της αντικατάστασης των ελεύθερων μεταβλητών του ερωτήματος με συγκεκριμένα ονόματα αντικειμένων $a_i$, γράφεται $q(\alpha_1, \dots, \alpha_n)$. Με βάση μια ανάθεση $\boldsymbol{\alpha}$, η ερμηνεία $\mathcal{I}$ συνεπάγεται τις εκφράσεις που συνθέτουν τα ερωτήματα όταν ισχύουν τα παρακάτω:

| | | |
|---|---|---|
| $\mathcal{I} \models^{\boldsymbol{\alpha}} C_j(t)$ | ανν | $t^{I,\boldsymbol{\alpha}} \in C_j^I$ |
| $\mathcal{I} \models^{\boldsymbol{\alpha}} R_j(t_{ja}, t_{jb})$ | ανν | $(t_{ja}, t_{jb}) \in R_j^I$ |
| $\mathcal{I} \models^{\boldsymbol{\alpha}} \varphi_1 \wedge \varphi_2$ | ανν | $\mathcal{I} \models^{\boldsymbol{\alpha}} \varphi_1$ και $\mathcal{I} \models^{\boldsymbol{\alpha}} \varphi_2$ |
| $\mathcal{I} \models^{\boldsymbol{\alpha}} \exists y_i \varphi$ | ανν | $\mathcal{I} \models^{\boldsymbol{\beta}} \varphi$ για μια ανάθεση $\boldsymbol{\beta}$ η οποία μπορεί να διαφέρει από την $\boldsymbol{\alpha}$ στην τιμή της $y_i$. |

Υπάρχουν και ερωτήματα που δεν περιέχουν μεταβλητές απάντησης[12]. Αυτά δεν επηρεάζονται από την αντιστοίχιση $\boldsymbol{\alpha}$ και έχουν ως απάντηση ένα «ναι» ή «όχι» που εξαρτάται αποκλειστικά από την $\mathcal{I}$. [10]

Η υπηρεσία της συνεπαγωγής ισχυρισμών μπορεί να θεωρηθεί ως μια ειδική περίπτωση του προβλήματος της απάντησης ερωτημάτων. Η απάντηση σε ένα ερώτημα μπορεί να δοθεί συνδυάζοντας με κατάλληλο τρόπο τις πληροφορίες που επιστρέφει η υπηρεσία συνεπαγωγής για ισχυρισμούς σχετικούς με το ερώτημα, ενδεχομένως με επανάληψη για κάθε σταθερά που περιλαμβάνεται στο ABox. Η διαδικασία μπορεί να απλοποιηθεί με την εισαγωγή στο TBox

[12] Τα ερωτήματα χωρίς μεταβλητές αποκαλούνται ground ή Boolean στην διεθνή βιβλιογραφία.

νέων εννοιών που προκύπτουν από το ερώτημα και τον έλεγχο των ισχυρισμών που ισχύουν για αυτές. Παρά την εγγύτητα με την συνεπαγωγή ισχυρισμών, η απάντηση συζευκτικών ερωτημάτων δεν μπορεί να αναχθεί στον έλεγχο συνέπειας [11].

## *2.5 Πολυπλοκότητα της συλλογιστικής*

Σε εφαρμογές όπου υπάρχουν εκτεταμένα σύνολα δεδομένων, όπως η διαχείριση ερωτημάτων που είδαμε στην προηγούμενη ενότητα ή η εκτέλεση ενός προγράμματος datalog σε βάση δεδομένων στην οποία θα αναφερθούμε στη συνέχεια, μπορούν να οριστούν τρεις τύποι πολυπλοκότητας. Τα δύο μεγέθη που εμφανίζονται ως είσοδος του προβλήματος είναι το «πρόγραμμα» και τα δεδομένα. Ο όρος «πρόγραμμα», ή καλύτερα «λογικό πρόγραμμα» μπορεί να αντιστοιχεί είτε σε ένα ερώτημα πάνω σε ένα TBox ή ένα σύνολο προτάσεων FOL είτε σε ένα πρόγραμμα datalog. Οι διαφορετικοί τύποι πολυπλοκότητας καθορίζονται από τα μεγέθη της εισόδου που θεωρούμε σταθερά. Με την πολυπλοκότητα δεδομένων (data complexity) εξετάζουμε τον τρόπο που επηρεάζεται η πολυπλοκότητα από διαφορετικά σύνολα δεδομένων, διατηρώντας σταθερό το «πρόγραμμα». Αντίθετα με την πολυπλοκότητα προγράμματος (program complexity) μελετάμε την επίδραση διαφορετικών «λογικών προγραμμάτων» πάνω σε σταθερό σύνολο δεδομένων. Τέλος, για την συνδυαστική πολυπλοκότητα (combined complexity) δεν θεωρούμε κάποιο από τα συστατικά του προβλήματος σταθερό.

Για να διασφαλιστεί η χρησιμότητα ενός συστήματος που βασίζεται σε ΠΛ οι υπηρεσίες συλλογιστικής που παρουσιάστηκαν προηγουμένως ή τουλάχιστον όσες από αυτές αφορούν την συγκεκριμένη εφαρμογή για την οποία προορίζεται, οφείλουν να είναι κατ' αρχήν αποφασίσιμες για την χρησιμοποιούμενη ΠΛ. Μόνο τότε είναι δυνατό να πραγματοποιηθεί ο συλλογισμός χωρίς την καθοδήγηση του χρήστη. Σε περιπτώσεις όπου είναι απαραίτητη μεγαλύτερη εκφραστικότητα από όση είναι διαθέσιμη από αποφασίσιμες ΠΛ, τότε είναι προτιμότερη η υλοποίηση της επιθυμητής συμπεριφοράς μέσω κωδικοποίησής της απευθείας στο πρόγραμμα, με τους περιορισμούς που αυτό συνεπάγεται ή η χρήση κάποιου άλλου φορμαλισμού αναπαράστασης γνώσης από την χρήση μιας μη αποφασίσιμης ΠΛ [12]. Η αντιμετώπιση αυτών των περιπτώσεων μπορεί για παράδειγμα να γίνει με συνδυασμό Answer Set Programming με ΠΛ [13], με την επέκταση των ΠΛ με κανόνες [14] ή τον συνδυασμό ΠΛ με datalog όπως συμβαίνει στα συστήματα AL-log [15] και CARIN [16].

Επιπλέον, είναι επιθυμητό οι υπηρεσίες συλλογιστικής να είναι χαμηλής πολυπλοκότητας για να επιλύονται ακόμα και σύνθετα προβλήματα σε εύλογο χρονικό διάστημα. Ωστόσο, η προσθήκη όλο και περισσότερων κατασκευαστών για την αύξηση της εκφραστικότητας οδηγεί μοιραία σε αύξηση της πολυπλοκότητας της διαδικασίας της συλλογιστικής. Δεν είναι τυχαίο το γεγονός ότι μεγάλο μέρος της έρευνας γύρω από τις ΠΛ έχει πέσει στην μελέτη της σχέσης της εκφραστικότητας με την πολυπλοκότητα. Οι διαθέσιμοι τελεστές αλληλεπιδρούν διαφορετικά μεταξύ τους με αποτέλεσμα η πολυπλοκότητα των διαφόρων ΠΛ να παρουσιάζει μεγάλες διακυμάνσεις. Καλό παράδειγμα των επιπτώσεων της αλληλεπίδρασης μεταξύ κατασκευαστών είναι η δραματική επιβάρυνση που επέρχεται από την ταυτόχρονη υποστήριξη της υπαγωγής ρόλων και περιορισμών πληθικότητας. Όπως αναφέρεται στο [10] για την βασική ΠΛ DL-Lite, η πολυπλοκότητα του ελέγχου της ικανοποιησιμότητας της βάσης γνώσης είναι NLogSpace πλήρης αν υποστηριχθεί μόνο ο ένας από τους δύο κατασκευαστές. Αν όμως χρησιμοποιηθούν στην ίδια βάση γνώσης ταυτόχρονα τότε η πολυπλοκότητα γίνεται ExpTime πλήρης. Μάλιστα η επιβάρυνση αυτή επέρχεται ακόμα και αν χρησιμοποιηθεί μόνο η απλούστερη μορφή περιορισμού πληθικότητας, ο συναρτησιακός περιορισμός. Το εμπόδιο αυτό οδήγησε στην δημιουργία μιας νέας ΠΛ, της DL-Lite$_A$ [17], η οποία περιορίζει καταλλήλως την αλληλεπίδραση των δύο κατασκευαστών αποφεύγοντας τις δυσάρεστες συνέπειες. Δεν είναι όμως πάντα βέβαιο ότι θα κληθούμε να πληρώσουμε την επέκταση μιας ΠΛ με αύξηση στην πολυπλοκότητα. Αν προστεθεί η υποστήριξη αξιωμάτων υπαγωγής ρόλων ή ο περιορισμός μεταβατικότητας ρόλων στην ΠΛ DL-Lite τότε δεν έχουμε κάποια μεταβολή στα αποτελέσματα που αφορούν στην πολυπλοκότητα της ικανοποιησιμότητας. Οι ονοματικές έννοιες (nominals) είναι επίσης ένας κατασκευαστής που μπορεί να έχει μεγάλη επίπτωση στην πολυπλοκότητα [18].

Πέρα από τους κατασκευαστές που παρέχονται από κάθε ΠΛ, η πολυπλοκότητα των υπηρεσιών συλλογιστικής μπορεί να επηρεάζεται και από το είδος του TBox. Τα απλά TBox είναι αυτά που επιβαρύνουν λιγότερο την διαδικασία της συλλογιστικής. Τα κυκλικά TBox δυνητικά προσθέτουν επιπλέον βάρος ενώ τα γενικευμένα είναι συνήθως τα πλέον επιβαρυντικά.

Στα προηγούμενα συνδέσαμε την αύξηση της εκφραστικότητας με την παροχή όλο και περισσότερων εργαλείων για την περιγραφή των καταστάσεων με όλο και μεγαλύτερη λεπτομέρεια. Υπάρχει όμως και μια άλλη ατραπός για την αύξηση της εκφραστικότητας που ίσως σε πρώτη ανάγνωση έρχεται σε αντίθεση με την διαίσθησή μας. Μέρος της έννοιας της εκφραστικότητας είναι και η δυνατότητα που έχουμε όταν περιγράφουμε μια κατάσταση να παραλείψουμε κάποια στοιχεία. Ενώ αρχικά μπορεί να φανεί σαν αδυναμία, το να είμαστε σε θέση να αφήσουμε ανοιχτά κάποια ενδεχόμενα μας επιτρέπει τελικά την περιγραφή περισσότερων πραγμάτων. Η καταγραφή της αοριστίας έρχεται με μεγάλο τίμημα που

δικαιολογείται εύκολα με ένα απλό παράδειγμα. Έστω ότι η βάση γνώσης αποτελείται από τις παρακάτω προτάσεις λογικής πρώτης τάξης:

μεγαλύτεροΒιβλίο ≠ βιβλίοΑ

μεγαλύτεροΒιβλίο = Αγαπημένο(Νίκου)

ΚαλύτεροΒιβλίο(βιβλίοΑ) ∨ ΚαλύτεροΒιβλίο(μεγαλύτεροΒιβλίο)

Υπάρχουν αρκετές αδιευκρίνιστες πληροφορίες. Αν το βιβλίοΑ δεν είναι το μεγαλύτερο τότε ποιο είναι; Το καλύτερο βιβλίο είναι το Α ή το πιο μεγάλο; Όλα αυτά τα ερωτήματα που αφήνονται ανοιχτά απαιτούν τη διερεύνηση πολλών περιπτώσεων. Μια πρόταση μπορεί να λειτουργεί πολλαπλασιαστικά στον αριθμό των περιπτώσεων που εισήγαγε κάποια άλλη, οπότε πολύ γρήγορα καταλήγουμε με ένα τεράστιο αριθμό περιπτώσεων προς διερεύνηση. Δεν είναι τυχαίο το γεγονός ότι η αοριστία της λογικής πρώτης τάξης περιορίζεται στις ΠΛ [19].

Αν οι χρησιμοποιούμενες υπηρεσίες συλλογιστικής έχουν τα επιθυμητά υπολογιστικά χαρακτηριστικά, η συμπεριφορά του συστήματος καθίσταται προβλέψιμη βελτιώνοντας την αξιοπιστία και κατά συνέπεια την καταλληλότητά του για μεγαλύτερο εύρος πρακτικών εφαρμογών. Για να επιτευχθούν όλοι αυτοί οι στόχοι είναι φανερό ότι πρέπει να περιοριστεί η εκφραστικότητα της χρησιμοποιούμενης ΠΛ στην απολύτως απαραίτητη για την περιγραφή των περιορισμών που απαιτεί το πεδίο ενδιαφέροντος. Ο περιορισμός της εκφραστικότητας απλοποιεί τη διαδικασία της συλλογιστικής και επιτρέπει την ανάπτυξη εξειδικευμένων τεχνικών, εφαρμόσιμων σε συγκεκριμένες ΠΛ όπως οι αλγόριθμοι tableaux ή η μέθοδος της μεταγραφής που θα δούμε σε επόμενο κεφάλαιο.

Ολοκληρώνοντας το κομμάτι της πολυπλοκότητας, αξίζει να αναφερθεί ότι ο περιορισμός της εκφραστικότητας δεν αποτελεί μονόδρομο για τη διατήρηση της πολυπλοκότητας σε διαχειρίσιμα επίπεδα. Οι άνθρωποι είναι σε θέση να πραγματοποιούν με ευκολία συλλογισμούς πάνω σε πληροφορίες που είναι κωδικοποιημένες στην φυσική γλώσσα, η οποία απέχει πολύ σε εκφραστικότητα από τις περιορισμένες γλώσσες των ΠΛ που εξετάζουμε εδώ. Το μυστικό ίσως βρίσκεται στο γεγονός ότι δεν εξετάζουν όλες τις λογικές συνέπειες της διαθέσιμης γνώσης ή μόνο τις συνέπειες αυτού του είδους. Για αυτό το λόγο, έχουν προταθεί και συστήματα μη πλήρους συλλογιστικής για την αντιμετώπιση ακραία εκφραστικών γλωσσών, στα οποία δεν θα αναφερθούμε στο παρόν κείμενο.

## *2.6 Περιγραφικές λογικές για απάντηση ερωτημάτων*

Όσον αφορά το πρόβλημα της σημασιολογικής ταξινόμησης δεδομένων, υπάρχουν δύο οικογένειες ΠΛ που παρουσιάζουν ιδιαίτερα καλή συμπεριφορά κατά την απάντηση ερωτημάτων. Η EL [20] και η DL-lite [21], στην οποία αναφερθήκαμε και νωρίτερα. Είναι πλέον σαφές ότι κατά την αναζήτηση ΠΛ με ιδιαίτερα ευνοϊκά υπολογιστικά χαρακτηριστικά είναι απαραίτητο να περιοριστούν οι διαθέσιμοι κατασκευαστές στους απολύτως απαραίτητους. Δεν είναι όμως απολύτως σαφές ποιο είναι το ελάχιστο σύνολο των απαραίτητων κατασκευαστών για να έχουμε έναν λειτουργικό φορμαλισμό αναπαράστασης γνώσης. Όπως αναφέρθηκε και στην εισαγωγή του κεφαλαίου, βασικό κίνητρο πίσω από τη δημιουργία των ΠΛ ήταν η ανάγκη απόδοσης τυπικής σημασιολογίας στα σημασιολογικά δίκτυα και τα πλαίσια (frames). Κατά την αποσαφήνιση της σημασιολογίας των θέσεων (slots) των πλαισίων και των ακμών ιδιοτήτων (property arcs) των σημασιολογικών δικτύων αποφασίστηκε ότι αντιστοιχούν σε περιορισμούς τιμής. Έτσι, θεωρήθηκε ως βασική η ΠΛ $FL_0$ η οποία επιτρέπει τον περιορισμό τιμής και την διάζευξη εννοιών μέσω του τελεστή ένωσης ⊔. Όμως, η παρουσία μη κενού TBox οδηγεί σε σημαντική πολυπλοκότητα. Ακόμα και αν έχουμε ένα απλό TBox η πολυπλοκότητα του ελέγχου της υπαγωγής εννοιών είναι coNP πλήρης [22]. Μεταβαίνοντας σε κυκλικά και γενικευμένα TBox η πολυπλοκότητα αυξάνεται σε PSPACE πλήρης [23] και EXPTIME πλήρης αντίστοιχα [24]. Χαμηλότερη πολυπλοκότητα φυσικά δεν μπορεί να παρατηρηθεί και για τις ΠΛ που επεκτείνουν την $FL_0$. Αρκετά χρόνια αργότερα η αναζήτηση υπολογιστικά βατών (tractable) ΠΛ οδήγησε στην αναθεώρηση της ανάγκης για υποστήριξη του περιορισμού τιμής. Έτσι ήρθε στο προσκήνιο η βασική ΠΛ EL, η οποία υποστηρίζει υπαρξιακό περιορισμό $\mathcal{E}$ και σύζευξη εννοιών μέσω του κατασκευαστή της τομής ⊓. Έχει αποδειχθεί ότι η υπαγωγή στην EL παραμένει υπολογιστικά βατή ακόμα και στην περίπτωση των γενικευμένων TBox [25]. Σύμφωνα με το [20] η πολυπλοκότητα διατηρείται στο ίδιο επίπεδο και στην περίπτωση επέκτασης της EL με διάφορους κατασκευαστές που είναι χρήσιμοι στις πρακτικές εφαρμογές. Εδώ θα μας απασχολήσει η ΠΛ ELHI που επεκτείνει την EL με την δυνατότητα δημιουργίας ιεραρχίας ρόλων και ορισμού αντίστροφων ρόλων. Συγκεκριμένα ένας ρόλος R στην ELHI μπορεί να είναι ο ατομικός ρόλος $P$ ή ο αντίστροφός του $P^-$. Μια ELHI έννοια μπορεί να λάβει τις εξής μορφές:

$$C \quad \rightarrow \quad \top \;\mid\; \bot \;\mid\; A \;\mid\; C_1 \sqcap C_2 \;\mid\; \exists R.C$$

Υπενθυμίζεται ότι η έννοια $A$ αποτελεί ατομική έννοια. Στο σώμα ορολογίας της ELHI μπορεί να περιέχονται αξιώματα γενικευμένων υπαγωγών εννοιών $C_1 \sqsubseteq C_2$ και αξιώματα υπαγωγής ρόλων $R_1 \sqsubseteq R_2$. Οι προσθήκες που είδαμε ανεβάζουν την πολυπλοκότητα του ελέγχου για την υπαγωγή εννοιών στην κλάση ExpTime [26]. Ωστόσο εδώ πρωτίστως ενδιαφερόμαστε για το πρόβλημα της απάντησης ερωτημάτων στο οποίο η ELHI παρουσιάζει πολύ καλή υπολογιστική συμπεριφορά διατηρώντας πολυωνυμική την πολυπλοκότητα δεδομένων [27]. Εδώ έχουμε ένα ακόμα παράδειγμα όπου η προσθήκη κατασκευαστών αφήνει ανεπηρέαστη τη δυσκολία των υπολογισμών, καθώς η ίδια πολυπλοκότητα παρατηρείται και για την βασική ΠΛ EL [28]. Έτσι εφαρμογές σημασιολογικής ταξινόμησης δεδομένων μπορούν να επωφεληθούν από την αύξηση στην εκφραστικότητα που προσφέρει η ELHI χωρίς επιβάρυνση της απόδοσής τους.

Η ετέρα οικογένεια ΠΛ που παρουσιάζει ενδιαφέροντα χαρακτηριστικά για την απάντηση ερωτημάτων είναι η DL-Lite. Ένας από τους στόχους κατά την ανάπτυξη της DL-Lite ήταν η εύκολη απάντηση ερωτημάτων πάνω σε μεγάλο όγκο δεδομένων που μπορεί να είναι αποθηκευμένα σε σχεσιακές βάσεις δεδομένων. Για να επιτευχθεί αυτός ο στόχος, η πολυπλοκότητα δεδομένων έπρεπε να διατηρηθεί ιδιαίτερα χαμηλά. Θα επικεντρωθούμε στην ΠΛ DL-Lite$_{R,\sqcap}$ που επεκτείνει την βασική ΠΛ DL-Lite με σύζευξη εννοιών στην αριστερή πλευρά των αξιωμάτων υπαγωγής και με προσοντούχο περιορισμό πληθικότητας στη δεξιά. Οι τύποι των εννοιών που μπορούν να περιγραφούν από την ΠΛ DL-Lite$_{R,\sqcap}$ είναι:

$$Cl \quad \rightarrow \quad A \mid \exists R \mid Cl_1 \sqcap Cl_2$$

$$Cr \quad \rightarrow \quad A \mid \exists R \mid \exists R.A \mid \neg A \mid \neg \exists R$$

Ο ρόλος $R$ μπορεί να είναι είτε ο ατομικός ρόλος $P$ είτε ο αντίστροφός του $P^-$. Το TBox μπορεί να περιέχει αξιώματα υπαγωγής εννοιών του τύπου $Cl \sqsubseteq Cr$ και ρόλων $R_1 \sqsubseteq R_2$. Παρά τη φαινομενική απλότητά της, η εκφραστικότητα που επιτυγχάνεται είναι επαρκής για να επιτρέπει την αναπαράσταση των βασικών χαρακτηριστικών φορμαλισμών εννοιολογικής μοντελοποίησης (conceptual data models) όπως το μοντέλο οντότητας σχέσης (Entity-Relationship) [29] και διαγραμμάτων κλάσεων UML [30]. Η πολυπλοκότητα δεδομένων για την απάντηση ερωτημάτων στην DL-Lite$_{R,\sqcap}$ είναι LogSpace. Αυτό σημαίνει ότι, σε αντίθεση με την ELHI, διατηρείται η ιδιότητα της επανεγγραψιμότητας πρώτης τάξης (FO rewritability), δηλαδή η δυνατότητα να παράγονται ισοδύναμα ερωτήματα στη γλώσσα SQL. Είναι συνεπώς δυνατή η απευθείας αξιοποίηση των υπαρχόντων και ιδιαίτερα ανεπτυγμένων συστημάτων σχεσιακών βάσεων δεδομένων.

## *2.7 Η γλώσσα OWL*

Το βασικότερο πρότυπο για την καταγραφή γνώσης είναι η γλώσσα περιγραφής οντολογιών OWL [31]. Η OWL προτάθηκε και εξελίσσεται από το W3C και τυγχάνει ευρείας αποδοχής. Ο όρος «οντολογία» προέρχεται από τη φιλοσοφία. Δεν υπάρχει κάποιος οικουμενικά αποδεκτός τυπικός ορισμός του όρου μεταξύ των επιστημόνων πληροφορικής αλλά σε κάθε περίπτωση χρησιμοποιείται για να περιγράψει ένα σώμα γνώσης. Λαμβάνοντας υπόψη τα όσα έχουν λεχθεί στο παρόν κείμενο, θα θεωρούμε μια οντολογία OWL ως μια βάση γνώσης γραμμένη με τρόπο που συμμορφώνεται πλήρως στις επιταγές του προτύπου. Το πρότυπο αυτή τη στιγμή βρίσκεται στην δεύτερη έκδοσή του και ορίζει μια σειρά διαφορετικών γλωσσών. Η OWL FULL παρέχει την μεγαλύτερη δυνατή εκφραστικότητα για την καταγραφή γνώσης. Η εκφραστικότητα αυτή έρχεται με βαρύ τίμημα καθώς οι βασικές υπηρεσίες συλλογιστικής καθίστανται μη αποφασίσιμες στην OWL FULL.

Η επόμενη σε σειρά εκφραστικότητας γλώσσα είναι η OWL DL. Απορρίπτει κάποιες από τις δυνατότητες έκφρασης της OWL FULL και καθίσταται αποφασίσιμη. Παραμένει βέβαια μια άκρως εκφραστική γλώσσα καθώς αντιστοιχεί στην ΠΛ SHOIN(**D**). Η πολυπλοκότητα της συλλογιστικής είναι υψηλή και συγκεκριμένα NExpTime πλήρης [18]. Μια ακόμα θετική συνέπεια του περιορισμού της εκφραστικότητας είναι η δυνατότητα γραφής της OWL DL και των επόμενων γλωσσών με την λεγόμενη αφηρημένη σύνταξη η οποία είναι πολύ πιο ξεκάθαρη και εύκολη στην χρήση από ανθρώπους. Η αυξημένη πολυπλοκότητα της OWL DL ανάγκασε τους δημιουργούς του προτύπου να ορίσουν τρία προφίλ, όπως τα ονομάζουν, δηλαδή γλώσσες που αποτελούν τμήματα της OWL DL με διαφορετική εκφραστικότητα και προσεκτικά σχεδιασμένες για να εξυπηρετούν τις απαιτήσεις ετερόκλητων εφαρμογών[13].

Η OWL QL (Query Language) δημιουργήθηκε με στόχο την διευκόλυνση της χρήσης οντολογιών για πρόσβαση σε ήδη υπάρχοντα συστήματα αποθήκευσης δεδομένων. Βασίζεται στην ΠΛ DL-Lite πράγμα που όπως είδαμε σημαίνει ότι η πολυπλοκότητα είναι πολυωνυμική ενώ διατηρεί τη δυνατότητα μεταγραφής πρώτης τάξης οπότε η απάντηση συζευκτικών ερωτημάτων μπορεί να εξυπηρετηθεί από σχεσιακές βάσεις χωρίς να απαιτείται κάποια αλλαγή στο επίπεδο των δεδομένων. Η OWL EL βασίζεται στην οικογένεια ΠΛ EL που

[13] Η πρώτη έκδοση της OWL όριζε την OWL Lite στη θέση των τριών προφίλ της δεύτερης έκδοσης. Η γλώσσα αυτή αντιστοιχεί στην ΠΛ SHIF(**D**) και η πολυπλοκότητά της είναι ExpTime. Η έλλειψη σημαντικά καλύτερων χαρακτηριστικών οδήγησε στη μη ανανέωσή της στην δεύτερη έκδοση του προτύπου. Λόγω της προς τα πίσω συμβατότητας παραμένει ως υπογλώσσα της OWL2.

επίσης είδαμε αναλυτικά. Το προφίλ αυτό στοχεύει σε εφαρμογές που απαιτούν ιδιαίτερα εκτεταμένες οντολογίες. Τα βασικά προβλήματα συλλογιστικής παραμένουν πολυωνυμικά [32]. Τέλος η OWL RL (Rule Language) σχεδιάστηκε με τέτοιο τρόπο ώστε κάθε οντολογία γραμμένη σε αυτήν να αντιστοιχεί σε ένα πρόγραμμα datalog. Τα προγράμματα datalog που δημιουργούνται από τη μετατροπή οντολογιών σε OWL QL περιέχουν κανόνες ενός σχετικά περιορισμένου είδους και μπορούν να αποτιμηθούν σε πολυωνυμικό χρόνο (συνδυασμένη πολυπλοκότητα) [33].

Ένας από τους βασικούς λόγους πίσω από την ευρεία αποδοχή του πρότυπου OWL είναι η διευκόλυνση που παρέχεται στους σχεδιαστές οντολογιών από τους έτοιμους μηχανισμούς συλλογιστικής για ΠΛ. Ο υπολογισμός της ταξινομίας των εννοιών μιας οντολογίας μπορεί να δώσει μια συνολική εποπτεία αυτής και να αναδείξει ακούσιες υπαγωγές που μπορεί να φανερώνουν λάθη στους ορισμούς. Εξ ίσου σημαντικός για την ανάδειξη λαθών είναι και ο έλεγχος ικανοποιησιμότητας των εννοιών. Το γεγονός ότι το πρότυπο της OWL βασίστηκε σε τέτοιο βαθμό στις ΠΛ αποτελεί ίσως την μεγαλύτερη αναγνώριση της αξίας των ΠΛ ως φορμαλισμού αναπαράστασης γνώσης [34].

Υπάρχουν και άλλες τεχνολογίες, πέρα των ΠΛ, που επηρέασαν τον σχεδιασμό της OWL. Η αφηρημένη σύνταξη της γλώσσας, η οποία είναι η πιο παραστατική και εύκολα κατανοητή από τον άνθρωπο διαμορφώθηκε από τα πλαίσια. Η απαίτηση για απόλυτη συμβατότητα με το προϋπάρχον πρότυπο RDF οδήγησε στην υιοθέτηση της RDF/XML σύνταξης [35].

## *2.8 Πρακτική υλοποίηση υπηρεσιών συλλογιστικής*

Ως γνωστόν, σε μία βάση γνώσης η πληροφορία δεν είναι στην ολότητά της αποθηκευμένη ρητά σε αυτήν αλλά μπορεί να απαιτηθεί να εκμαιευτεί νέα για να απαντηθεί κάποιο ερώτημα. Με την άντληση πληροφορίας να είναι ο κύριος λόγος ύπαρξης των βάσεων γνώσης το σύστημα πίσω από την διαδικασία αυτή, η μηχανή συλλογιστικής (reasoning engine), αποτελεί ένα από τα πιο σημαντικά μέρη. Η μηχανή συλλογιστικής είναι ένα πρόγραμμα που συνάγει τις λογικές συνεπαγωγές από τα αξιώματα του σώματος ορολογίας και τα γεγονότα που περιέχονται στο σώμα ισχυρισμών. Η εξαγωγή συμπερασμάτων αποτελεί ένα υπολογιστικά δύσκολο πρόβλημα. Δύο είναι οι κύριοι παράγοντες που μπορεί να επιβαρύνουν την διαδικασία αυτή, η εκφραστικότητα της χρησιμοποιούμενης οντολογίας και τα ιδιαίτερα εκτεταμένα σύνολα δεδομένων που μπορεί να την συνοδεύουν. Στην πλειονότητα των περιπτώσεων, μία απάντηση που έρχεται μετά από ώρες επεξεργασίας δεν

είναι πρακτικά χρήσιμη. Η απαίτηση η πληροφορία να είναι όσο το δυνατόν πιο άμεσα διαθέσιμη έχει οδηγήσει στην επινόηση πολλών διαφορετικών τεχνικών για την αξιοποίηση της γνώσης που κωδικοποιείται σε βάσεις αυτού του είδους.

Πέρα από τις πολλές διαφορετικές τεχνικές με τις οποίες μπορεί να υλοποιηθεί μια μηχανή συλλογιστικής, υπάρχει όπως είδαμε πληθώρα υπηρεσιών συλλογιστικής που είναι δυνατό να προσφέρονται. Σε επίπεδο οντολογίας οι κυριότερες είναι ο έλεγχος συνέπειας και η ταξινόμηση. Άλλες είναι η ικανοποιησιμότητα, η υπαγωγή ή ισοδυναμία εννοιών καθώς και ο έλεγχος για ξένες μεταξύ τους έννοιες. Όσον αφορά το σώμα ισχυρισμών, οι κύριες υπηρεσίες που μπορεί να παρέχονται είναι ο έλεγχος συνέπειας και η αναζήτηση συνεπαγωγών, λαμβάνοντας υπόψη και το σώμα ορολογίας. Οι συνεπαγωγές αυτές αφορούν τις σταθερές που περιλαμβάνονται στο σώμα ισχυρισμών και μέσω αυτών διευκρινίζεται αν συμμετέχουν σε κάποιο ρόλο ή αποτελούν μέλος κάποιας έννοιας του σώματος ορολογίας. Εκτός των βασικών υπηρεσιών συλλογιστικής πάνω στο σώμα ισχυρισμών, σε ορισμένες υλοποιήσεις παρέχεται η δυνατότητα ανάκτησης όλων των ατόμων που ανήκουν σε κάποια έννοια ή των εννοιών στις οποίες ανήκει ένα άτομο και όμοια για ρόλους, η ανάκτηση των ρόλων στους οποίους μετέχει ένα άτομο ή τα άτομα με τα οποία συνδέεται. Ένας απλός τρόπος υλοποίησης αυτών των υπηρεσιών είναι ο διαδοχικός έλεγχος όλων των σταθερών που εμφανίζονται στο σώμα ισχυρισμών [36]. Επιπλέον, υπηρεσίες που μπορεί να παρέχονται, σε συνδυασμό με τις υπηρεσίες συλλογιστικής, είναι η επεξήγηση του τρόπου με τον οποίο αποδεικνύεται κάποιο συμπέρασμα ή βοήθεια για την αποσφαλμάτωση της βάσης κατά τη φάση ανάπτυξης. Όπως είδαμε στην ενότητα για τις περιγραφικές λογικές, αν υποστηρίζεται άρνηση πολλές υπηρεσίες συλλογιστικής είναι θεωρητικά δυνατό να αναχθούν στο πρόβλημα της ασυνέπειας της βάσης γνώσης. Όμως στην πράξη, αυτή η προσέγγιση δεν αποδίδει πάντα ικανοποιητικά με αποτέλεσμα να είναι απαραίτητη η εφαρμογή εξειδικευμένων βελτιστοποιήσεων για κάθε υπηρεσία [37]. Για αυτό το λόγο δεν πρέπει να θεωρείται δεδομένη η υποστήριξη όλων των υπηρεσιών από κάθε υλοποίηση.

Υπάρχει πληθώρα υλοποιήσεων συστημάτων συλλογιστικής με τις διαφορές μεταξύ τους να εκτείνονται από τον τρόπο που αντιμετωπίζουν το δίλλημα μεταξύ εκφραστικότητας και επεξεργαστικής πολυπλοκότητας, τις παρεχόμενες υπηρεσίες συλλογιστικής και τις δυνατότητες χειρισμού ιδιαίτερα εκτεταμένων οντολογιών ή συνόλων δεδομένων. Για παράδειγμα, οι υλοποιήσεις που βασίζονται σε αλγόριθμους tableaux είναι συνήθως σε θέση να αντιμετωπίζουν αποτελεσματικά μεγάλες και εκφραστικές οντολογίες, όπως αυτές που συναντάμε στο πεδίο της βιολογίας. Εκ διαμέτρου αντίθετα χαρακτηριστικά συναντάμε σε συστήματα που σχεδιαστικά βρίσκονται κοντά στις βάσεις δεδομένων τα οποία προχωρούν από την αρχή στην εξαγωγή όλων των συμπερασμάτων και την αποθήκευσή τους. Αυτά τα συστήματα είναι σε θέση να χειριστούν αποτελεσματικά μεγάλα σύνολα γεγονότων αλλά

στην πλειονότητά τους υποστηρίζουν περιορισμένης εκφραστικότητας οντολογίες. Στη συνέχεια παρουσιάζονται οι διάφορες κατηγορίες συστημάτων συλλογιστικής και οι κυριότερες υλοποιήσεις αυτών.

### *2.8.1 Αλγόριθμοι tableaux*

Οι αλγόριθμοι tableaux [38] παρέχουν τη δυνατότητα απόδειξης της ικανοποιησιμότητας ή μη κάποιας έκφρασης μέσω της σταδιακής απλοποίησής της. Για να το επιτύχουν αυτό, χρησιμοποιούν ένα σύνολο από «κανόνες επέκτασης» (tableaux expansion rules) που τροποποιούνται ανάλογα με τους κατασκευαστές που επιτρέπονται στην γλώσσα που χρησιμοποιείται για την κωδικοποίηση της οντολογίας. Για αυτό το λόγο είναι πολύ εύκολο να προσαρμοστούν για χρήση με μια μεγάλη γκάμα γλωσσών απλώς επιλέγοντας διαφορετικούς κανόνες επέκτασης [39]. Για να αποφανθεί για τη συνέπεια μιας βάσης γνώσης ο αλγόριθμος προσπαθεί να κατασκευάσει ένα μοντέλο αυτής αποδομώντας σταδιακά τις περιγραφές εννοιών που περιέχει χρησιμοποιώντας τους κανόνες επέκτασης. Αυτή η διαδικασία οδηγεί στην εκμαίευση όλο και περισσότερων υπονοούμενων περιορισμών στα δυνατά μοντέλα. Δύο είναι τα ενδεχόμενα. Είτε να ολοκληρωθεί η διαδικασία οδηγώντας σε ένα μοντέλο της βάσης είτε η συνεχής προσθήκη περιορισμών να καταλήξει σε κάποια προφανή αντίφαση που κάνει αδύνατη την δημιουργία οποιουδήποτε μοντέλου.

Τα συστήματα που χρησιμοποιούν αλγόριθμους tableaux είναι εξαιρετικά για συλλογιστική πάνω σε εκφραστικά και εκτεταμένα σώματα ορολογίας [37]. Μάλιστα έχει αποδειχθεί ότι οι αλγόριθμοι αυτοί είναι βέλτιστοι για μια σειρά περιγραφικών λογικών καθώς η πολυπλοκότητά τους στη χειρότερη περίπτωση ταυτίζεται με αυτήν του προβλήματος ικανοποιησιμότητας για τις λογικές αυτές [40].

Εκτός από την απάντηση στο πρόβλημα της ικανοποιησιμότητας, οι αλγόριθμοι tableaux παρέχουν και ένα χρήσιμο υποπροϊόν. Ο γράφος που δημιουργείται για τον υπολογισμό της απάντησης μπορεί να χρησιμοποιηθεί εκ νέου για την αποδοτικότερη υλοποίηση υπηρεσιών συλλογιστικής ανώτερου επιπέδου [41].

Παρουσιάζουν ωστόσο αδυναμία να χειριστούν αποδοτικά περιπτώσεις που υπάρχουν μεγάλα σύνολα δεδομένων [42]. Ο λόγος για αυτό βρίσκεται στην κεντρική ιδέα πίσω από την υλοποίηση των αλγόριθμων tableaux που απαιτεί την ξεχωριστή εξέταση κάθε πρότασης προς απόδειξη. Όταν για παράδειγμα πρέπει να διερευνηθεί ποιες σταθερές του σώματος ισχυρισμών αποτελούν μέρος της ερμηνείας κάποιας έννοιας C απαιτείται ο έλεγχος της συνέπειας των υπαρχόντων ισχυρισμών με την προσθήκη του ισχυρισμού $a: \neg C$, μία φορά

για κάθε σταθερά $a$. Αν δεν προκύψει αντίφαση η σταθερά $a$ υπάγεται στην ερμηνεία της έννοιας $C$. Η εκτέλεση του αλγόριθμου μία φορά για κάθε σταθερά ευθύνεται για την δυσκολία χειρισμού μεγάλων σωμάτων ισχυρισμών. Επιπλέον, είναι δύσκολο να αποτραπεί η διερεύνηση ασύνδετων με το ερώτημα συνεπαγωγών. Τέλος, η μη ντετερμινιστική τους φύση αποτελεί ένα επιπλέον εμπόδιο στην ανάπτυξη γρήγορων υλοποιήσεων [43]. Παρά τις βελτιστοποιήσεις που έχουν παρουσιαστεί [44] και την ανάπτυξη των αλγόριθμων hypertableaux [45] η δυσκολία στην συλλογιστική πάνω σε μεγάλα σώματα ισχυρισμών παραμένει.

Συστήματα συλλογιστικής που χρησιμοποιούν αλγόριθμους tableaux είναι τα παρακάτω:

- FaCT++: Υποστηρίζει την περιγραφική λογική SHOIQ(D−). Αποτελεί την επόμενη γενιά του FaCT [46]. Επανακωδικοποιήθηκε σε C++ για καλύτερη απόδοση ενώ ενσωματώθηκαν αρκετές επιπλέον βελτιώσεις [47].
- RacerPro: Πρόκειται για ένα σύστημα που ενσωματώνει τις βελτιστοποιήσεις του FaCT και είναι γραμμένο σε Common Lisp. Ο αλγόριθμος που χρησιμοποιείται είναι σχεδιασμένος για την περιγραφική λογική SHIQ [48]. Υποστηρίζει πολλαπλά σώματα ορολογίας και ισχυρισμών, πραγματικούς αριθμούς και ορισμένες πολυωνυμικές εξισώσεις με αυτούς και τέλος, έλεγχο ισότητας συμβολοσειρών. Ενσωματώνει την γλώσσα nQRL [49] μέσω της οποίας παρέχονται προηγμένες υπηρεσίες συλλογιστικής πάνω στο σώμα ισχυρισμών. Αποτελεί το τελευταίο μέρος της οικογένειας που ξεκίνησε από το RACE [50] και συνεχίστηκε με το RACER [51] ενώ πλέον έχει ξεκινήσει να διατίθεται και εμπορικά [52]. Επιπλέον παρέχεται και δυνατότητα συλλογιστικής με ονοματικές έννοιες η οποία όμως δεν είναι πλήρης. [51]
- HermiT: Στο HermiT επιτυγχάνεται υψηλή απόδοση με τη χρήση αλγόριθμων hypertableaux. Οι αλγόριθμοι αυτοί αντιμετωπίζουν το πρόβλημα της έντονης μη ντετερμινιστικής φύσης των απλών tableaux περιορίζοντας τον υπολογιστικό φόρτο. Υποστηρίζει την γλώσσα OWL 2 DL και είναι γραμμένο σε Java. [53]
- Pellet: Το πρώτο σύστημα συλλογιστικής που υποστήριξε πλήρως την OWL-DL (SHOIN(D)) και έκτοτε έχει επεκταθεί και υποστηρίζει την OWL 2 (SROIQ(D)) [32]. Είναι εξολοκλήρου γραμμένο σε Java ενώ παρέχει και υποστήριξη κανόνων [54]. Σε συνδυασμό με κάποια μηχανή datalog μπορούν να απαντηθούν ερωτήματα διατυπωμένα σε AL-log [15]. Πέρα από τις συνήθεις υπηρεσίες συλλογιστικής στο σώμα ορολογίας περιλαμβάνει δυνατότητες που αφορούν στο σώμα ισχυρισμών, όπως ο χειρισμός συζευκτικών ερωτημάτων (conjunctive queries) μέσω των γλωσσών SPARQL και RDQL. [55]

### *2.8.2 Μηχανές κανόνων*

Η κατηγορία αυτή δυνητικά περιλαμβάνει μια ευρεία και σχετικά ανομοιογενή ομάδα συστημάτων. Η έννοια του κανόνα, της λογικής συνεπαγωγής με τις διάφορες παραλλαγές της είναι στοιχειώδης για την αναπαράσταση γνώσης. Για αυτό το λόγο, μπορεί κανείς να συμπεριλάβει στην συγκεκριμένη κατηγορία το σύνολο σχεδόν των συστημάτων γνώσης. Όμως ακόμα και αν αγνοήσουμε την συμβολή των κανόνων στην καταγραφή της γνώσης, τα όρια της κατηγορίας παραμένουν ασαφή. Για παράδειγμα τα HermiT, Pellet και RacerPro που παρουσιάστηκαν στην προηγούμενη κατηγορία υποστηρίζουν τη γλώσσα κανόνων SWRL. Η κατηγοριοποίηση δικαιολογείται αφού αποτελούν υλοποιήσεις αλγορίθμων tableaux. Επιπλέον, η Prolog και κατ' επέκταση η datalog, μπορούν να θεωρηθούν γλώσσες κανόνων. Όμως τα συστήματα που αξιοποιούν τεχνικές μεταγραφής δίνοντας το τελικό αποτέλεσμα σε αυτές τις γλώσσες δεν θα συμπεριληφθούν σε αυτή την κατηγορία καθώς η διαδικασία της μεταγραφής αποτελεί το κυριότερο χαρακτηριστικό τους. Τα συστήματα που θα αναφερθούν εδώ αποτελούνται από μηχανές κανόνων και δεν περιλαμβάνουν χαρακτηριστικά που θα επέτρεπαν την ένταξή τους σε κάποια άλλη κατηγορία. Σε αυτά, η δομή της γνώσης κωδικοποιείται σε ένα σύνολο κανόνων τους οποίους στη συνέχεια χρησιμοποιεί η μηχανή κανόνων για πραγματοποιήσει τους απαραίτητους συλλογισμούς. Παρακάτω παρουσιάζονται τα κυριότερα συστήματα που καλύπτουν τις διαφορετικές πτυχές της πολυσχιδούς αυτής κατηγορίας.

- Owlim: Παρέχεται σαν επέκταση της βάσης τριάδων RDF Sesame [56] και είναι γραμμένο σε Java και C. Το Sesame παρέχει μόνο βασικές υπηρεσίες εξαγωγής συμπερασμάτων για RDF(S). Ανάλογα με το σύνολο κανόνων που θα επιλεγούν, το Owlim [57] επεκτείνει τις δυνατότητες συλλογιστικής παρέχοντας υποστήριξη μέχρι το επίπεδο της OWL-RL και ένα υποσύνολο της OWL-Lite, το αποκαλούμενο OWL-Horst [58], που μπορεί να αντιμετωπιστεί με κανόνες. Βασίζεται στην μηχανή κανόνων Triple Reasoning and Rule Entailment Engine, TRREE που ακολουθεί την από κάτω προς τα πάνω προσέγγιση. Πραγματοποιεί πλήρη εκμαίευση της υπονοούμενης πληροφορίας κατά την φόρτωση του συνόλου δεδομένων για πιο άμεση ανταπόκριση στα ερωτήματα. Διάδοχο του Owlim αποτελεί το σύστημα GraphDB.
- O-Device: Βασίζεται στην μηχανή κανόνων παραγωγής CLIPS[14]. Οι συντακτικές δομές της OWL μετατρέπονται σε αντικείμενα πάνω στα οποία επιδρούν οι κανόνες. Υποστηρίζεται η OWL-Lite και ορισμένα χαρακτηριστικά της OWL-DL. Το O-

[14] http://www.clipsrules.net

Device [59] δεν υποστηρίζει αποθήκευση σε δευτερεύουσες μονάδες μνήμης πράγμα που περιορίζει την δυνατότητα χειρισμού εκτεταμένου συνόλου ισχυρισμών. Ακόμα ένα σύστημα διαχείρισης κανόνων που μπορεί να πραγματοποιήσει συλλογισμούς πάνω σε οντολογίες είναι το Jrools[15]. Με το λογισμικό που παρουσιάζεται στο [60] μεταφράζει ορισμένες από τις δομές της OWL στην κατάλληλη μορφή για την εισαγωγή στο Jrools. Λόγω των περιορισμών που υπάρχουν η λύση αυτή μάλλον δεν μπορεί να θεωρηθεί ολοκληρωμένη και απευθύνεται σε υπάρχοντες χρήστες του Jrools που επιθυμούν την εκμετάλλευση οντολογιών. Και τα δύο συστήματα χρησιμοποιούν τον αλγόριθμο Rete [61] στις μηχανές κανόνων τους.

- FLORA-2: Εσωτερικά χρησιμοποιεί την γλώσσα F-Logic [62] που επεκτείνει τις δυνατότητες μοντελοποίησης των γλωσσών λογικού προγραμματισμού συνδυάζοντάς τες με την λογική των πλαισίων επιτυγχάνοντας έτσι την εισαγωγή στοιχείων αντικειμενοστραφούς σχεδιασμού. Το FLORA-2 [63] που χρησιμοποιεί την μηχανή XSB [64]. Η XSB, όντας σύστημα Prolog, αποτιμά το ερώτημα από πάνω προς τα κάτω. Σε αντίθεση με την απλή Prolog η XSB ενσωματώνει τον μηχανισμό του «tabling» [65] ο οποίος, πέρα από βελτιώσεις στην διαχείριση ερωτημάτων, προσδίδει χαρακτηριστικά που εμφανίζουν οι μηχανές που ακολουθούν την αντίθετη κατεύθυνση κατά την διαδικασία του συλλογισμού και έχει ως αποτέλεσμα τον τερματισμό σε περιπτώσεις που η αποτίμηση σε Prolog δεν θα τερμάτιζε. Άλλο ένα σύστημα που αποτελεί υλοποίηση της F-Logic είναι τα FLORID [66].
- Jena[16]: Αποτελεί μία συλλογή εργαλείων για την δημιουργία εφαρμογών για τον σημασιολογικό ιστό. Παρέχει μια μηχανή κανόνων, με διάφορα σύνολα κανόνων, για συλλογιστική επιπέδου RDFS [67]. Συλλογιστική υψηλότερου επιπέδου μπορεί να επιτευχθεί με τη χρήση τρίτων μηχανών εξαγωγής συμπερασμάτων (πχ Pellet).
- RDFox: Αποτελεί μια βάση τριάδων RDF σχεδιασμένη να εκμεταλλεύεται πολυπύρηνα συστήματα που λειτουργεί αποκλειστικά στην κύρια μνήμη. Είναι γραμμένη σε C++ αλλά παρέχει δυνατότητα χρήσης και από εφαρμογές σε Java και Python. Υποστηρίζει συλλογιστική στο επίπεδο OWL-RL. Η φόρτωση οντολογιών σε OWL2 γίνεται μέσω του OWL API[17], στην συνέχεια το τμήμα τους που αντιστοιχεί στην OWL-RL μεταφράζεται σε κανόνες datalog για να εκμαιευτούν οι υπονοούμενες τριάδες. [68]

[15] https://www-01.ibm.com/software/integration/business-rule-management/jrules-family/

[16] http://jena.apache.org/

[17] http://owlapi.sourceforge.net

### *2.8.3 Συλλογιστική με χρήση αποδείκτη θεωρημάτων*

Δεν πρόκειται για μια προσέγγιση στο πρόβλημα της συλλογιστικής καθώς τα συστήματα απόδειξης θεωρημάτων που χρησιμοποιούν έχουν σχεδιαστεί με διαφορετικές απαιτήσεις και για πολύ διαφορετική χρήση και ως εκ τούτου δεν εντυπωσιάζουν με την απόδοσή τους.

- Hoowlet[18]: Εδώ χρησιμοποιείται μία εντελώς διαφορετική τεχνική [69] για συλλογιστική σε OWL-DL. Η οντολογία μεταφράζεται σε αξιώματα λογικής πρώτης τάξης τα οποία στη συνέχεια δίνονται στον αποδείκτη θεωρημάτων Vampire [70] για έλεγχο συνέπειας. Παρόλο που ο Vampire είναι ένα από τα ταχύτερα συστήματα για τη λογική πρώτης τάξης, οι επιδόσεις του στο τμήμα της OWL-DL δεν εντυπωσιάζουν σε σύγκριση με τα εξειδικευμένα συστήματα συλλογιστικής. Πολύ σημαντικό ρόλο παίζει η μετάφραση με τις επιδόσεις να διαφοροποιούνται σημαντικά για διαφορετικές μεταφράσεις.

### *2.8.4 Η τεχνική της μεταγραφής*

Χρησιμοποιώντας την τεχνική της μεταγραφής μπορούμε να αντιμετωπίσουμε το πρόβλημα του χειρισμού του μεγάλου όγκου δεδομένων με ώριμες και δοκιμασμένες τεχνικές, όπως αυτές των επαγωγικών βάσεων δεδομένων, που έχουν αναπτυχθεί και βελτιστοποιηθεί ακριβώς για αυτό τον σκοπό. Οι βάσεις αυτές επεξεργάζονται τις πληροφορίες ως σύνολο και δεν απαιτείται η εκτέλεση κάποιας λειτουργίας ξεχωριστά για κάθε ισχυρισμό. Επιπλέον, ενσωματώνουν τεχνικές όπως τα μαγικά σύνολα που επιτρέπουν την εστίαση της διαδικασίας συλλογισμού μόνο στο τμήμα του ABox που είναι σχετικό με το ερώτημα. Επειδή η διαδικασία χειρισμού του ABox είναι ντετερμινιστική, είναι δυνατή η αποτελεσματική δεικτοδότηση των δομών στη βάση πράγμα που οδηγεί σε ακόμα ταχύτερη αποτίμηση των ερωτημάτων επιτρέποντας την αποτελεσματική χρήση εκτεταμένων σωμάτων ισχυρισμών σε πρακτικές εφαρμογές. Ειδικά στις περιπτώσεις όπου η μεταγραφή μπορεί να γίνει σε απλή datalog τα οφέλη είναι τεράστια αφού η πολυπλοκότητα δεδομένων της γλώσσας αυτής είναι πολυωνυμική, πράγμα που διασφαλίζει την δυνατότητα ανάπτυξης μιας πολύ ικανοποιητικής υλοποίησης. Σύμφωνα με την παρούσα βιβλιογραφία η περιγραφική λογική Horn-SHIQ είναι πιθανά η πλέον εκφραστική ΠΛ της οποίας οι οντολογίες είναι δυνατό να μεταγραφούν σε απλή datalog. Προσπάθειες για την επέκταση αυτού του ορίου σε μη Horn-SHIQ οντολογίες

[18] http://owl.man.ac.uk/hoolet/

έχουν στεφθεί με μερική επιτυχία καθώς δεν έχουν αποδώσει καθολικά εφαρμόσιμες τεχνικές [33].

Η μεταγραφή λειτουργεί ως εξής: Το ερώτημα και η οντολογία αποτελούν την είσοδο η οποία μεταγράφεται σε ένα σύνολο προτάσεων, συνήθως πρόκειται για ένα λογικό πρόγραμμα σε κάποια διάλεκτο της datalog όμως υπάρχουν αρκετές εναλλακτικές όπως θα δούμε παρακάτω. Ο ρόλος της οντολογίας τελειώνει εδώ καθώς οι περιορισμοί που περιλαμβάνει κωδικοποιούνται στο πρόγραμμα. Το πρόγραμμα αυτό μπορεί στη συνέχεια να εκτελεστεί πάνω στο όποιο σύνολο δεδομένων μπορεί να συνοδεύσει την οντολογία και να δώσει την σωστή απάντηση στο αρχικό ερώτημα. Αυτό που ουσιαστικά επιτυγχάνεται με την μεταγραφή είναι η απομάκρυνση της επίδρασης του σώματος ισχυρισμών από την κύρια διαδικασία συλλογιστικής κατά την οποία λαμβάνεται υπόψη μόνο το σώμα ορολογίας και το ερώτημα.

Καλύτερη αίσθηση για το τι ακριβώς είναι η μεταγραφή ερωτημάτων μπορεί να αποκτηθεί μέσω της συνοπτικής παρουσίασης ενός σχετικά απλού αλγόριθμου μεταγραφής για την περιγραφική λογική DL-Lite$_A$. Πρόκειται για τον αλγόριθμο PerfectRef που παρουσιάζεται αναλυτικά στο [71]. Παίρνει ως είσοδο ένα συζευκτικό ερώτημα (CQ) q και ένα TBox σε DL-Lite$_A$ και επιστρέφει την μεταγραφή του σαν μία ένωση από συζευκτικά ερωτήματα (Union of Conjunctive Queries – UCQ).

### Παράδειγμα 1: PerfectRef.

TBox: $T = \{C \sqsubseteq \exists R, R \sqsubseteq S\}$, ABox: $A = \{C(a)\}$

ερώτημα: $q(x, y) \leftarrow R(x, z), S(y, z)$

Ο αλγόριθμος, όπως και οι περισσότεροι του είδους του, εφαρμόζουν κανόνες ανάλυσης με σκοπό να επεκτείνουν το αρχικό ερώτημα. Εδώ εφαρμόζεται ο κανόνας AtomRewrite που αξιοποιεί κάθε (θετικό) αξίωμα υπαγωγής για την επέκταση των διαθέσιμων CQ. Στο παράδειγμά μας ο AtomRewrite θα χρησιμοποιήσει το αξίωμα $R \sqsubseteq S$ και θα επιστρέψει:

UCQ: { $q(x, y) \leftarrow R(x, z), S(y, z)$, $q(x, y) \leftarrow R(x, z), R(y, z)$ }

Στις περιπτώσεις που αντικαθίσταται ρόλος από έννοια για την διατήρηση της ορθότητας πρέπει η μεταβλητή που εξαφανίζεται να μην εμφανίζεται πουθενά αλλού στο ερώτημα.

Αν εκτελέσουμε το ερώτημα ως έχει δεν θα επιστραφεί κάποια απάντηση. Η αντικατάσταση ατόμων δεν είναι αρκετή. Πρέπει να εφαρμοστεί και ένας δεύτερος κανόνας, ο Reduce, ο οποίος εφαρμόζεται πάνω σε κάθε CQ που έχει δύο άτομα τα οποία μπορούν να ενοποιηθούν επιστρέφοντας ένα νέο CQ. Έτσι από το $q(x, y) \leftarrow R(x, z), R(y, z)$ θα προκύψει το $q(x, x) \leftarrow R(x, z)$. Τέλος ο AtomRewrite χρησιμοποιώντας το $C \sqsubseteq \exists R$ θα δώσει την τελική μεταγραφή:

UCQ: { $q(x,y) \leftarrow R(x,z), S(y,z)$, $q(x,y) \leftarrow R(x,z), R(y,z)$,
$q(x,x) \leftarrow R(x,z)$, $q(x,x) \leftarrow C(x)$. }

Κατά τη χρήση μιας οντολογίας είναι σύνηθες να τίθενται πολύπλοκα ερωτήματα για τις σταθερές του σώματος ισχυρισμών, τα οποία φυσικά για να απαντηθούν απαιτείται να αξιοποιηθεί η πληροφορία που βρίσκεται στο σώμα ορολογίας. Σε πρακτικές εφαρμογές αναμένεται το μέγεθος του σώματος ισχυρισμών να είναι κατά πολύ μεγαλύτερο από αυτό του σώματος ορολογίας. Για αυτό το λόγο, όταν διερευνούμε την πολυπλοκότητα της διαδικασίας απάντησης ερωτημάτων κυρίαρχο ρόλο παίζει η έκταση του σώματος ισχυρισμών δηλαδή η πολυπλοκότητα δεδομένων.

Όπως είναι αναμενόμενο, η εκφραστικότητα της ΠΛ στην οποία είναι γραμμένο το TBox επηρεάζει και την πολυπλοκότητα της διαδικασίας απάντησης ερωτημάτων και κατ' επέκταση την γλώσσα στην οποία μπορεί να κωδικοποιηθεί η μεταγραφή. Η πλέον απλή περίπτωση είναι αυτή της DL-Lite. Όπως αποδεικνύεται στο [21] η μεταγραφή ερωτημάτων στην DL-Lite μπορεί να αποδοθεί σαν ένα σύνολο συζευκτικών ερωτημάτων με το τελικό αποτέλεσμα να λαμβάνεται από την ένωση των απαντήσεων σε αυτά. Η πολυπλοκότητα δεδομένων για αυτή την κατηγορία ερωτημάτων είναι στο LogSpace πράγμα που σημαίνει ότι μπορούν να αποτιμηθούν απευθείας πάνω στο ABox από ένα σύστημα διαχείρισης σχεσιακής βάσης δεδομένων (FO rewritability). Τα αυτά ισχύουν και για την DL-Lite $_{F,\cap}$ που αποτελεί την επέκταση της DL-Lite με σύζευξη εννοιών στο αριστερό μέλος των υπαγωγών εννοιών [72]. Αν χρησιμοποιήσουμε πιο εκφραστικές περιγραφικές λογικές όπως η horn-SHIQ, η πολυπλοκότητα της αναζήτησης των στοιχείων του ABox που ικανοποιούν τους περιορισμούς των ερωτημάτων γίνεται PTime πλήρης [73]. Η αποτίμηση αυτών των ερωτημάτων απαιτεί την πλήρη ισχύ της datalog. Τέλος υπάρχουν περιγραφικές λογικές όπου η απάντηση ερωτημάτων πάνω στο ABox εμπίπτει στην κατηγορία co-NP. Σε αυτές τις περιπτώσεις για την απάντηση των ερωτημάτων απαιτείται (τουλάχιστον) η διαζευκτική datalog. Ένα παράδειγμα γλώσσας αυτής της κατηγορίας είναι η πλήρης SHIQ [43]. Η διαζευκτική datalog είναι μια επέκταση της datalog όπου επιτρέπονται οι διαζεύξεις στην κεφαλή των κανόνων. Είναι μια ισχυρή γλώσσα που μπορεί να μοντελοποιήσει αόριστες καταστάσεις. Η εκφραστική δύναμη έρχεται με σημαντικό κόστος αφού η συνδυασμένη πολυπλοκότητα είναι co-NEXPTIME πλήρης ενώ η πολυπλοκότητα δεδομένων είναι co-NP πλήρης [74]. Η αυξημένη πολυπλοκότητα της διαζευκτικής datalog καθιστά πολύ δύσκολη την ανάπτυξη συστημάτων με δυνατότητες χειρισμού μεγάλου όγκου δεδομένων.

Στη συνέχεια θα αναφερθούν συστήματα που επιτρέπουν την συλλογιστική πάνω σε εκτεταμένα σύνολα δεδομένων μεταγράφοντας το αρχικό ερώτημα σε κάποιον διαφορετικό φορμαλισμό που διευκολύνει την απάντησή του.

#### *2.8.4.1 Συστήματα συλλογιστικής για την οικογένεια περιγραφικών λογικών DL-Lite*

Όπως είδαμε σε προηγούμενο κεφάλαιο η DL-Lite είναι μια οικογένεια περιγραφικών λογικών οι οποίες είναι σχεδιασμένες με γνώμονα την εύκολη πρόσβαση σε μεγάλου όγκου δεδομένα μέσω οντολογιών. Για αυτού του είδους τα TBox τα ερωτήματα μεταγράφονται σε UCQ το μέγεθος των οποίων όμως στην χειρότερη περίπτωση μπορεί να είναι εκθετικά μεγαλύτερο από το αρχικό ερώτημα. Το ίδιο ισχύει και αν επιτρέψουμε τη μεταγραφή σε πιο εκφραστικούς φορμαλισμούς όπως ερωτήματα κωδικοποιημένα σε λογική πρώτης τάξης [75]. Το γεγονός αυτό δεν αποτελεί απαγορευτικό μειονέκτημα αφού το μέγεθος του ερωτήματος είναι συνήθως περιορισμένο. Τα συστήματα που θα αναφερθούν παρακάτω αντιμετωπίζουν με διάφορους τρόπους το πρόβλημα περιορίζοντας στο μέτρο του δυνατού το μέγεθος της μεταγραφής.

- Presto: Εδώ το πρόβλημα του εκθετικού μεγέθους της μεταγραφής αντιμετωπίζεται με τον υπολογισμό της σαν μη αναδρομικό πρόγραμμα datalog το οποίο στο τέλος μεταφράζεται σε SQL. Επιπλέον οι κανόνες επέκτασης του ερωτήματος είναι βελτιωμένοι σε σχέση με τις προηγούμενες προσπάθειες και έτσι η μεταγραφή που παράγεται δεν είναι εκθετικά μεγαλύτερη σε σύγκριση με τα άτομα στο αρχικό ερώτημα αλλά σε σχέση με κάποιες μεταβλητές που συμμετέχουν σε συγκεκριμένου τύπου συνενώσεις στο σώμα του ερωτήματος. Πέρα από το μέγεθος, μειώνεται και ο χρόνος υπολογισμού της μεταγραφής ενώ φαίνεται να υπάρχει πλεονέκτημα και κατά την αποτίμηση του προγράμματος datalog [76]. Εξέλιξη του συστήματος αυτού είναι το Prexto το οποίο ενώ βασίζεται στον ίδιο αλγόριθμο εφαρμόζει επιπλέον βελτιώσεις πετυχαίνοντας περαιτέρω μείωση στο μέγεθος της μεταγραφής [77].
- QuOnto: Το σύστημα αυτό μετατρέπει τα CQ που αποτελούν την μεταγραφή σε εντολές SQL οι οποίες αποτιμώνται απευθείας στην σχεσιακή βάση [78]. Αποτελεί την καρδιά του Mastro [79] το οποίο αποτελεί ένα ολοκληρωμένο σύστημα διαχείρισης δεδομένων μέσω οντολογιών (OBDA – Ontology Based Data Access).
- Ontop: Πρόκειται για ένα πλήρες σύστημα ODBA που υποστηρίζει οντολογίες εκφραστικότητας OWL2-QL, αποθηκευμένες σε σχεσιακές βάσεις δεδομένων. Για τη βελτίωση της μεταγραφής εστιάζει σε δύο πηγές πολυπλοκότητας που βρίσκονται πίσω από την εκθετική αύξηση του μεγέθους της μεταγραφής. Η πρώτη είναι ότι στο TBox μπορεί να ορίζεται μεγάλος αριθμός υποεννοιών και υπορόλων οι οποίοι πρέπει να περιλαμβάνονται στη μεταγραφή αν στο αρχικό ερώτημα υπάρχουν οι πιο γενικές έννοιες και ρόλοι. Η δεύτερη κρύβεται πίσω από πολλαπλούς ορισμούς για όρους της οντολογίας που μπορεί να βρίσκονται στην αντιστοίχιση με το σχήμα της σχεσιακής βάσης και επηρεάζει όταν τα UCQ μετατρέπονται σε εντολές SQL [80].

- IQAROS: Το πρωτότυπο αυτό σύστημα είναι σε θέση να επεκτείνει την υπάρχουσα μεταγραφή ενός ερωτήματος καθώς ο χρήστης εξειδικεύει την αναζήτησή του [81].
- Quest: Ένα αρκετά ολοκληρωμένο σύστημα που υποστηρίζει ερωτήματα σε SPARQL[19] και συλλογιστική σε OWL-DL. Μπορεί να διαχειριστεί ABox, τα οποία βρίσκονται σε παλαιού τύπου βάσεις δεδομένων μέσω δοσμένων αντιστοιχίσεων με τους όρους του TBox [82].
- Nyaya: Άλλο ένα πρωτότυπο σύστημα το οποίο επιδεικνύει τον αλγόριθμο TGD-rewrite [83]. Χρησιμοποιεί την μηχανή datalog IRIS η οποία έχει επεκταθεί για να είναι σε θέση να υποστηρίξει την μεταγραφή που προκύπτει.

#### *2.8.4.2 Συστήματα που υποστηρίζουν πιο εκφραστικές περιγραφικές λογικές*

- Requiem: Το Requiem υποστηρίζει μια σειρά περιγραφικών λογικών με τον αλγόριθμό του να κλιμακώνεται ανάλογα σε πολυπλοκότητα και να παραμένει βέλτιστος στην χειρότερη περίπτωση, όσον αφορά στην πολυπλοκότητα δεδομένων, για κάθε μία από αυτές. Πιο συγκεκριμένα όταν το TBox είναι δοσμένο σε EL, ELH, ELHI τότε η μεταγραφή είναι ένα πρόγραμμα datalog. Στην περίπτωση της DL-Lite$^{+}$[20] επιστρέφει ένα UCQ συνοδευόμενο από ένα ερώτημα σε γραμμική datalog ενώ για DL-Lite$_{R}$ ή DL-Lite$_{core}$[21] η μεταγραφή αποτελείται μόνο από το UCQ [84].
- Rapid: Υποστηρίζει σώματα ορολογίας γραμμένα σε DL-Lite και ELHI με σχέδια για μελλοντική επέκταση σε ακόμα πιο εκφραστικές περιγραφικές λογικές. Στην περίπτωση της DL-Lite ο αλγόριθμος που χρησιμοποιεί επιτυγχάνει υψηλές επιδόσεις αποτρέποντας την δημιουργία συζευκτικών ερωτημάτων των οποίων η απάντηση δεν προσφέρει επιπλέον πληροφορία σε σχέση με άλλα, πιο γενικά, που βρίσκονται στην μεταγραφή κάνοντας περιττό τον έλεγχο για τέτοια ερωτήματα στο τέλος της διαδικασίας. Το Rapid θα το εξετάσουμε με μεγαλύτερη λεπτομέρεια αμέσως μετά.

[19] Βλ. κεφάλαιο για RDF.

[20] Η DL-Lite$^{+}$ προκύπτει από την ELHI απαγορεύοντας αντίστροφους ρόλους και αξιώματα της μορφής $A \sqcap B \sqsubseteq C$.

[21] Περισσότερα για τα είδη των γλωσσών της οικογένειας DL-Lite στο [10].

- Clipper: Το Clipper υποστηρίζει μέχρι την περιγραφική λογική Horn-SHIQ. Το TBox τροποποιείται σύμφωνα με συγκεκριμένους κανόνες ανάλυσης και στη συνέχεια το ερώτημα μεταγράφεται με βάση το επεξεργασμένο TBox σε πρόγραμμα datalog [85].
- KAON2: Είναι ένα από τα πλέον εκφραστικά συστήματα αυτής της κατηγορίας με τη διαδικασία της μεταγραφής να επεκτείνεται στην πλήρη έκταση της περιγραφικής λογικής SHIQ καθώς και στο DL-safe[22] τμήμα της SWRL [86]. Το KAON2 είναι γραμμένο σε Java και τα κυριότερα σημεία της αρχιτεκτονικής του είναι τα εξής: Η μηχανή κατηγοριοποίησης της οντολογίας, που αναλαμβάνει την μετάφραση του TBox σε όσο γίνεται λιγότερες προτάσεις λογικής πρώτης τάξης. Ο αριθμός των προτάσεων παίζει πολύ σημαντικό ρόλο στην συνολική απόδοση. Το υποσύστημα απόδειξης θεωρημάτων που αναλαμβάνει να επεξεργαστεί τους κανόνες και να δημιουργήσει το πρόγραμμα datalog. Τέλος η μηχανή datalog η οποία τρέχει το πρόγραμμα και δίνει την τελική απάντηση [42]. Η τεχνολογία πίσω από το KAON2 έχει αξιοποιηθεί εμπορικά μέσω του OntoBroker [87] το οποίο περιλαμβάνει ορισμένες βελτιώσεις όπως την δυνατότητα αποτελεσματικής εκμετάλλευσης συστημάτων με πολλούς επεξεργαστές.
- DLog: Πρόκειται για ένα ακόμα σύστημα συλλογιστικής σε ABox βασισμένο στην επεξεργασία του TBox από κανόνες ανάλυσης. Υποστηρίζει και αυτό την πλήρη περιγραφική λογική SHIQ με την διαφορά ότι η μεταγραφή που προκύπτει είναι στην γλώσσα Prolog, με την ίδια γλώσσα να χρησιμοποιείται και στην ανάπτυξη του ίδιου του DLog [88]. Αρχικά τα αξιώματα του TBox μετατρέπονται σε προτάσεις λογικής πρώτης τάξης που υπακούουν σε ορισμένους περιορισμούς και στη συνέχεια με βάση αυτές δημιουργείται το πρόγραμμα Prolog. Προτέρημα του συστήματος αποτελεί η εστιασμένη αναζήτηση στο μέρος του ABox που είναι σχετικό με το ερώτημα.

#### *2.8.4.3 Rapid*

Το σύστημα για το οποίο κατασκευάστηκε το API είναι το Rapid. Αυτή την στιγμή υποστηρίζει μεταγραφή για ερωτήματα πάνω σε οντολογίες κατασκευασμένες με τις

[22] Οι κανόνες αυτού του είδους είναι συντακτικά περιορισμένοι για να είναι εφαρμόσιμοι μόνο σε άτομα που αναφέρονται στο σώμα ισχυρισμών. Με αυτό τον τρόπο αποφεύγεται η απώλεια αποφασισιμότητας.

περιγραφικές λογικές DL-Lite$_{R,\cap}$ (στη συνέχεια θα παραλείπονται οι δείκτες) και ELHI. Οι συγκεκριμένες περιγραφικές λογικές έχουν βαρύνουσα σημασία καθώς όπως είδαμε στο προηγούμενο κεφάλαιο αποτελούν τη βάση των προφίλ OWL QL και OWL EL της γλώσσας OWL 2 που αποτελεί την πρόταση του W3C για την περιγραφή οντολογιών για χρήση στον σημασιολογικό ιστό. Η κεντρική ιδέα πίσω από τον αλγόριθμο που υλοποιεί το Rapid προέρχεται από το πεδίο της μαθηματικής λογικής και της απόδειξης θεωρημάτων. Χρησιμοποιείται λογισμός βασισμένος στην ανάλυση (resolution based calculi) που έχει το πλεονέκτημα ότι επιδεικνύει βέλτιστη απόδοση στην χειρότερη περίπτωση[23] και επιτρέπει την εφαρμογή πολλών βελτιστοποιήσεων που προέρχονται από την έρευνα στα παραπάνω πεδία.

Τα κύρια σημεία της δημιουργίας της μεταγραφής είναι τα εξής:

### *A) Clausification*

Αρχικά το σώμα ορολογίας της βάσης γνώσης στην οποία απευθύνεται το ερώτημα μετατρέπεται σε ένα σύνολο προτάσεων Horn. Πιο συγκεκριμένα, αυτό γίνεται ακολουθώντας την ισοδυναμία των αξιωμάτων των περιγραφικών λογικών και των προτάσεων της λογικής πρώτης τάξης όπως αυτή φαίνεται, για τις περιγραφικές λογικές που υποστηρίζει το Rapid, στον πίνακα 2. Σε αυτό το σημείο πρέπει να αναφερθεί ότι τα αξιώματα στην αριστερή στήλη του πίνακα 2 αντιστοιχούν σε κανονικοποιημένα TBox εκφρασμένα σε DL-Lite$_{R,\cap}$ (στη συνέχεια θα παραλείπονται οι δείκτες) ή ELHI. Τα αξιώματα στις τελευταίες δύο γραμμές ονομάζονται αξιώματα RA και δεν αποτελούν μέρος της DL-Lite ενώ τα υπόλοιπα είναι κοινά και για τις δύο ΠΛ. Όπου εμφανίζεται πλήρης υπαρξιακός περιορισμός στο δεξί μέρος του αξιώματος αντικαθίσταται με συναρτησιακό σύμβολο, ένα για κάθε εμφάνιση τέτοιου περιορισμού. Αυτή η αντικατάσταση αντικατοπτρίζει τη διαδικασία του skolemization για τη συγκεκριμένη μεταβλητή. Κάθε τέτοιο σύμβολο υποδηλώνει μια συνάρτηση που αντιστοιχεί μία σταθερά σε κάθε τιμή που μπορεί να πάρει η ανεξάρτητη μεταβλητή. Για παράδειγμα το αξίωμα $A \subseteq \exists R.B$ αντιστοιχεί σε δύο προτάσεις. Η πρώτη πρόταση, $R(x, f(x)) \leftarrow A(x)$, δείχνει ότι το $A(x)$ συνεπάγεται την ύπαρξη μιας εξαρτημένης σταθεράς $f(x)$. Η $f(x)$ εξαρτάται από τη σταθερά που θα μπει στην θέση της μεταβλητής x και μετέχει στην σχέση $R$ με αυτή. Η δεύτερη, $B(f(x)) \leftarrow A(x)$, προσδιορίζει τον τύπο της $f(x)$.

[23] Όσο αφορά στην πολυπλοκότητα δεδομένων.

**Πίνακας 2: Αντιστοιχία αξιωμάτων ELHI και προτάσεων λογικής πρώτης τάξης.**

| Αξίωμα σώματος ορολογίας | Πρόταση FOL |
|---|---|
| $B \subseteq A$ | $A(x) \leftarrow B(x)$ |
| $B \cap C \subseteq A$ | $A(x) \leftarrow B(x) \wedge C(x)$ |
| $\exists R \subseteq A$ | $A(x) \leftarrow R(x,y)$ |
| $\exists R^{-} \subseteq A$ | $A(x) \leftarrow R(y,x)$ |
| $P \subseteq R$ | $R(x,y) \leftarrow P(x,y)$ |
| $P \subseteq R^{-}$ | $R(x,y) \leftarrow P(y,x)$ |
| $A \subseteq \exists R.B$ | $R(x,f(x)) \leftarrow A(x)$<br>$B(f(x)) \leftarrow A(x)$ |
| $A \subseteq \exists R^{-}.B$ | $R(f(x),x) \leftarrow A(x)$<br>$B(f(x)) \leftarrow A(x)$ |
| $\exists R.C \subseteq A$ | $A(x) \leftarrow R(x,y) \wedge C(y)$ |
| $\exists R^{-}.C \subseteq A$ | $A(x) \leftarrow R(y,x) \wedge C(y)$ |

Στη συνέχεια χρησιμοποιούνται κανόνες εξαγωγής συμπερασμάτων (inference rules) ειδικά κατασκευασμένοι για τις ιδιαιτερότητες της κάθε υποστηριζόμενης περιγραφικής λογικής. Οι κανόνες αυτοί αποτυπώνονται με την μορφή $\frac{A \ B_1..B_n}{\Gamma\sigma}$ όπου το $A$ είναι η κύρια υπόθεση (main premise), με $B$ αναπαριστώνται μία ή περισσότερες δευτερεύουσες υποθέσεις (side premises) και $\Gamma$ το συμπέρασμα (resolvent) που επάγεται από τις υποθέσεις. Το σ (unifier) συμβολίζει τις αντιστοιχίες μεταξύ των μεταβλητών της κύριας υπόθεσης και των δευτερευόντων με τις οποίες επιτυγχάνεται η ενοποίηση τους. Οι κανόνες εξαγωγής συμπερασμάτων που αποτελούν τον λογισμό (calculi) του Rapid διακρίνονται σε τρία είδη, ανάλογα με το στάδιο του αλγόριθμου στο οποίο εμφανίζονται.

## *B)Unfolding*

Για την DL-Lite έχουμε τον κανόνα $\frac{Q \ C}{Q'\sigma}$ όπου το Q είναι το ερώτημα ή κάποια τροποποιημένη εκδοχή του που προέκυψε ως συμπέρασμα από προηγούμενη εφαρμογή του κανόνα, το C συμβολίζει μια από τις προτάσεις DL-Lite του ABox, το Q΄ είναι το παραγόμενο συμπέρασμα και το σ συμβολίζει την αντιστοιχία των μεταβλητών. Κατάλληλοι περιορισμοί στις δυνατές τιμές του σ διασφαλίζουν ότι το συμπέρασμα Q΄ δεν περιέχει συναρτησιακά σύμβολα. Αυτοί οι περιορισμοί παίζουν πολύ σημαντικό ρόλο στην βελτίωση των επιδόσεων του Rapid αφού διασφαλίζουν ότι δεν σπαταλούνται πόροι για τον υπολογισμό εκφράσεων που δεν θα συμπεριληφθούν στην μεταγραφή.

Στην περίπτωση της ELHI ο κανόνας τροποποιείται ελαφρώς επιτρέποντας τη χρήση επιπλέον προτάσεων πέρα από το ερώτημα σαν κύριες υποθέσεις. Συμβολίζοντας με Υ κάποια από τις προτάσεις που αντιστοιχούν είτε στα επιπλέον αξιώματα της ELHI που δεν εμφανίζονται στην DL-Lite (αξιώματα RA, τελευταίες δύο γραμμές του πίνακα 2) είτε σε ερώτημα, ο κανόνας γράφεται $\frac{\Upsilon\ C}{\Upsilon'\sigma}$ με το συμπέρασμα Υ΄ να παραμένει ελεύθερο από συναρτησιακά σύμβολα.

*Γ) Shrinking*

Εδώ περιλαμβάνονται σύνθετοι κανόνες εξαγωγής συμπερασμάτων με περισσότερες από μία δευτερεύουσες υποθέσεις. Ξεκινάμε με μία πρόταση, που δεν περιέχει συναρτησιακά σύμβολα, η οποία σε συνδυασμό με την πρώτη δευτερεύουσα υπόθεση δίνει ως συμπέρασμα μια πρόταση $\Gamma_1$ η οποία περιλαμβάνει κάποιο συναρτησιακό σύμβολο. Στη συνέχεια με την διαδοχική εφαρμογή των υπόλοιπων δευτερευουσών υποθέσεων επιχειρείται η εξάλειψη αυτού του συναρτησιακού συμβόλου. Έτσι η αλυσίδα των συμπερασμάτων $\Gamma_1 \dots \Gamma_n, \Gamma'$ καταλήγει στο $\Gamma'$ το οποίο δεν περιέχει συναρτησιακά σύμβολα και έτσι μπορεί να αποτελέσει μέρος της μεταγραφής.

Όπως και στο προηγούμενο βήμα έτσι και εδώ ο κανόνας για την DL-Lite μπορεί να έχει σαν κύρια υπόθεση μόνο το ερώτημα $Q$ ή συμπεράσματα που έχουν εξαχθεί προηγουμένως και γράφεται $\frac{Q\ \ C_1\ [C_2]}{Q'}$ . Οι αγκύλες γύρω από το $C_2$ υπονοούν ότι είναι προαιρετικό. Για να εισαχθεί συναρτησιακό σύμβολο στο ενδιάμεσο συμπέρασμα $Q_1$ πρέπει αυτό να αναφέρεται στο $C_1$. Για να εξαλειφθεί στη συνέχεια από το $Q_1$ και να πάρουμε το τελικό συμπέρασμα πρέπει το ίδιο συναρτησιακό σύμβολο να αναφέρεται και στο $C_2$. Όπως βλέπουμε στον πίνακα 2, οι μόνες δυνατές δευτερεύουσες υποθέσεις για αυτό τον κανόνα είναι της μορφής $R(x, f(x)) \leftarrow A(x)$ και $B(f(x)) \leftarrow A(x)$.

**Παράδειγμα 2: Εξάλειψη συναρτησιακού συμβόλου.**

$Q_A : Q(x) \leftarrow R(x, y) \wedge B(y)$

Δευτερεύουσες υποθέσεις:

(1): $R\big(x, f(x)\big) \leftarrow A(x)$ (2): $B(f(x)) \leftarrow A(x)$

Shrinking: $\frac{Q_A\ (1)\ (2)}{Q(x) \leftarrow A(x)} : \frac{Q_A\ (1)}{Q(x) \leftarrow A(x) \wedge B(f(x))}$ & $\frac{Q(x) \leftarrow A(x) \wedge B(f(x))\ (2)}{Q(x) \leftarrow A(x)}$

Όπως και στην προηγούμενη κατηγορία κανόνων για την ELHI θα χρησιμοποιήσουμε πέρα από τα ερωτήματα και τα αξιώματα RA σαν κύριες υποθέσεις. Όμως εδώ έχουμε μία ακόμα διαφορά σε σχέση με την DL-Lite. Ο κανόνας μπορεί να επεκταθεί περαιτέρω και να περιλάβει περισσότερες δευτερεύουσες υποθέσεις. Έτσι έχουμε τον κανόνα n-shrinking: $\frac{\Upsilon\ C_1\ [C_2 \dots C_n]}{\Upsilon'}$. Το συμπέρασμα $\Upsilon'$παραμένει ελεύθερο από συναρτησιακά σύμβολα. Ο κανόνας αυτός θα εμφανιστεί με περισσότερες από δύο δευτερεύουσες υποθέσεις μόνο στην περίπτωση που χρειαστεί να εκτελεστεί κάποιος κανόνας της επόμενης κατηγορίας.

### *Δ) Function*

Στην κατηγορία αυτή δεν περιλαμβάνονται κανόνες για την DL-Lite. Οι κανόνες που θα δούμε εδώ χρησιμοποιούνται όταν υπάρχει ταυτόχρονα κάποια πρόταση που περιέχει ένα ρόλο με έναν από τους όρους του να είναι κάποιο συναρτησιακό σύμβολο και η αντίστροφη πρόταση από το clausification κάποιου αξιώματος RA. Συγκεκριμένα οι κανόνες αυτής της κατηγορίας είναι της μορφής $\frac{B(x) \leftarrow R(x,y) \wedge [C_1(y) \dots C_n(y)] \qquad R(f(x),x) \leftarrow A(x)}{B(f(x)) \leftarrow A(x) \wedge [C_1(y) \dots C_n(y)]}$ και $\frac{B(x) \leftarrow R(y,x) \wedge [C_1(y) \dots C_n(y)] \qquad R(x,f(x)) \leftarrow A(x)}{B(f(x)) \leftarrow A(x) \wedge [C_1(x) \dots C_n(x)]}$. Όπως βλέπουμε παράγουν προτάσεις με συναρτησιακά σύμβολα οι οποίες μπορούν να αξιοποιηθούν σαν επιπλέον δευτερεύουσες υποθέσεις στους κανόνες n-shrinking και να οδηγήσουν στην παραγωγή χρήσιμων προτάσεων που θα συμπεριληφθούν στη μεταγραφή. Το πλήθος των κανόνων σε αυτή την κατηγορία είναι άμεσα συνδεδεμένο με τον αριθμό των αντίστροφων ρόλων που υπάρχουν στο σώμα ορολογίας. Αν δεν έχουμε αντίστροφους ρόλους δεν είναι δυνατό να παραχθεί ως συμπέρασμα πρόταση που να αναφέρει συναρτησιακό σύμβολο και ο κανόνας n-shrinking θα έχει πάντα μέχρι δύο δευτερεύουσες υποθέσεις όπως συμβαίνει και στην περίπτωση της DL-Lite.

Όπως γίνεται αντιληπτό η παραπάνω διαδικασία κλιμακώνεται με ιδανικό τρόπο όσο προστίθεται περαιτέρω εκφραστικότητα στο σώμα ισχυρισμών. Η απλούστερη δυνατή περίπτωση είναι αυτή της DL-Lite όπου οι κανόνες εξαγωγής συμπερασμάτων έχουν σαν κύριες υποθέσεις μόνο το αρχικό ερώτημα και όποια συμπεράσματα προκύπτουν στην συνέχεια. Αν έχουμε να κάνουμε με σώμα ισχυρισμών εκφρασμένο στην ELH, που δεν περιέχει αντίστροφους ρόλους, τότε οι κανόνες εξαγωγής συμπερασμάτων διατηρούν την ίδια μορφή απλώς προστίθενται τα αξιώματα RA σαν επιλογή για κύριες υποθέσεις. Τέλος αν υπάρχουν και αντίστροφοι ρόλοι τότε μπορούν να εφαρμοστούν και κανόνες function σε συνδυασμό με κανόνες n-shrinking με περισσότερες από δύο δευτερεύουσες υποθέσεις. Αυτή η σταδιακή κλιμάκωση είναι ένα ιδιαίτερα επιθυμητό χαρακτηριστικό αφού ο όγκος του

υπολογιστικού έργου είναι ανάλογος της πολυπλοκότητας της περιγραφικής λογικής και των χαρακτηριστικών που χρησιμοποιούνται στο σώμα ορολογίας.

Επιπροσθέτως, πρέπει να αναφερθεί ότι οι ιδιαίτερα αυξημένες επιδόσεις του Rapid οφείλονται και στην έξυπνη υλοποίηση πέρα από την καλά σχεδιασμένη διαδικασία ανάλυσης. Δεν υπολογίζονται πλήρως τα ενδιάμεσα αποτελέσματα με συνέπεια την μεγάλη εξοικονόμηση πόρων. Εκμεταλλεύεται όπου είναι δυνατό τις ομοιότητες μεταξύ των προτάσεων και με τις κατάλληλες μετονομασίες μεταβλητών επιτυγχάνεται η επαναχρησιμοποίηση έτοιμων αποτελεσμάτων. Τέλος γίνεται έγκαιρη αναγνώριση και απόρριψη των συζευκτικών ερωτημάτων τα οποία έχουν ως απάντηση υποσύνολο της απάντησης άλλων ερωτημάτων που θα συμπεριληφθούν στη μεταγραφή. Έτσι παράγεται μικρότερη μεταγραφή που διευκολύνει τη διαδικασία της αποτίμησής.

# 3

# *Datalog*

Η datalog είναι μία δηλωτική γλώσσα που ανήκει στο πεδίο του λογικού προγραμματισμού [89]. Ο όρος δηλωτική σημαίνει ότι, σε αντίθεση με τις προστακτικές γλώσσες, δεν καθορίζει την διαδικασία με την οποία θα υπολογιστεί το επιθυμητό αποτέλεσμα παρά μόνο το περιγράφει. Ο λογικός προγραμματισμός αναπτύχθηκε πάνω στις βάσεις που έθεσε η έρευνα για την αυτοματοποιημένη απόδειξη θεωρημάτων [90]. Κινητήριος δύναμη ήταν η πεποίθηση ότι η γλώσσα της λογικής πρώτης τάξης ήταν ιδανική και για την αναπαράσταση της γνώσης και για την περιγραφή των αποτελεσμάτων που αναζητούνται [91]. Η datalog είναι μία από τις κατ' εξοχήν γλώσσες για την δημιουργία επαγωγικών βάσεων δεδομένων (deductive databases). Η τοποθέτηση της γλώσσας αυτής ανάμεσα στον λογικό προγραμματισμό και τις βάσεις δεδομένων είναι εμφανής και από το όνομά που αποτελεί σύνθεση των όρων data και logic [92]. Αποτελεί υποσύνολο της γλώσσας προγραμματισμού Prolog [93]. Η ανάγκη ορισμού μιας διαφορετικής γλώσσας για την υποστήριξη των επαγωγικών βάσεων δεδομένων προκύπτει από τη διαπίστωση ότι το ερώτημα αν κάποιο γεγονός υπονοείται από ένα πρόγραμμα με την πλήρη εκφραστικότητα της Prolog είναι μη αποφασίσιμο. Η δυσκολία αυτή είναι απόρροια του γεγονότος ότι η Prolog αποτελεί μια πλήρη γλώσσα προγραμματισμού. Κατά συνέπεια η ύπαρξη κάποιας αυτοματοποιημένης διαδικασίας που να δίνει απάντηση στο παραπάνω ερώτημα θα σήμαινε ότι θα ήμασταν σε θέση να λύσουμε το πρόβλημα τερματισμού. Η δυσκολία αυτή ξεπερνιέται δίνοντας στον χρήστη της Prolog έλεγχο πάνω στην διαδικασία εξαγωγής συμπερασμάτων [94]. Η ανάγκη για καθοδήγηση από τον χρήστη απαιτεί εξειδικευμένες γνώσεις και εμπειρία από μέρους του. Όμως η εισαγωγή

μιας τέτοιας απαίτησης θα περιόριζε δραματικά την χρησιμότητα των συστημάτων αναπαράστασης γνώσης και των βάσεων γνώσης. Για να γίνει αποφασίσιμη η διαδικασία της εκμαίευσης γεγονότων απαιτείται η μείωση της εκφραστικότητας της χρησιμοποιούμενης γλώσσας. Για αυτό το λόγο η datalog επιτρέπει την χρήση μεταβλητών και σταθερών μόνο, αποκλείεται δηλαδή η χρήση συναρτήσεων με ένα ή περισσότερα ορίσματα[24]. Με αυτό τον τρόπο επιτυγχάνεται η επιθυμητή υπολογιστική συμπεριφορά. Δηλαδή το ερώτημα αν το γεγονός Γ υπονοείται από ένα πρόγραμμα datalog P το οποίο τρέχει πάνω στα γεγονότα D που συνήθως βρίσκονται σε μια βάση δεδομένων, $(P \cup D) \vDash \Gamma$, μπορεί πάντα να απαντηθεί από μία πλήρως αυτοματοποιημένη υπολογιστική διαδικασία.

Υπάρχουν δύο χρήσιμες ιδιότητες που παραμένουν μη αποφασίσιμες για τα προγράμματα datalog παρά τον περιορισμό στην εκφραστικότητα. Η πρώτη είναι το αν ένα πρόγραμμα περιέχεται σε κάποιο άλλο. Αν έχουμε δύο προγράμματα Α και Β που παράγουν πληροφορία για κάποιο κοινό κατηγόρημα Τ και η πληροφορία που παράγει το Α για το Τ είναι υποσύνολο αυτής που παράγει το Β για το Τ για κάθε δυνατή είσοδο τότε λέμε ότι το Α περιέχεται στο Β ως προς το κατηγόρημα Τ. Αυτή η ιδιότητα θα ήταν πολύ χρήσιμη στην βελτιστοποίηση των προγραμμάτων datalog. Το δεύτερο ερώτημα είναι αν το βάθος της αναδρομής σε κάποιο πρόγραμμα περιορίζονται σε ένα μέγιστο άνω όριο που ισχύει για κάθε διαφορετική είσοδο. Προγράμματα που περιορίζονται κατά αυτό τον τρόπο παύουν ουσιαστικά να είναι αναδρομικά πράγμα που διευκολύνει τον χειρισμό τους.

Στην περίπτωση της datalog όπου το πρόγραμμα συνοδεύεται από ένα, συχνά εκτεταμένο, σύνολο αρχικών γεγονότων μπορούν να οριστούν οι τρεις τύποι πολυπλοκότητας που είδαμε στο κεφάλαιο των ΠΛ. Η πολυπλοκότητα δεδομένων είναι πολυωνυμικού χρόνου (P) και η πολυπλοκότητα προγράμματος είναι εκθετικού χρόνου (ExpTime). Η συνδυαστική πολυπλοκότητα αποδεικνύεται ισοδύναμη με την πολυπλοκότητα προγράμματος αφού το ένα πρόβλημα μπορεί να αναχθεί στο άλλο μέσω μετατροπών πολυωνυμικού χρόνου [95]. Πρωτίστως ενδιαφέρει η πολυπλοκότητα δεδομένων καθώς το μέγεθος του προγράμματος είναι στην πλειονότητα των περιπτώσεων αισθητά μικρότερο από αυτό του συνόλου δεδομένων (γεγονότα).

[24] Οι σταθερές θεωρούνται συναρτήσεις χωρίς κάποιο όρισμα.

## *3.1 Συντακτικό*

Ένα πρόγραμμα datalog αποτελείται από ένα πεπερασμένο σύνολο γεγονότων και κανόνων. Τα γεγονότα είναι ισχυρισμοί για τον κόσμο που μοντελοποιείται και οι κανόνες είναι προτάσεις μέσω των οποίων μπορούμε να εξάγουμε νέα γεγονότα βασιζόμενοι σε υπάρχοντα. Ένα παράδειγμα γεγονότος είναι η πρόταση «ο Νίκος δουλεύει στην εταιρία com». Η πρόταση «τα διευθυντικά στελέχη της com είναι οι μάνατζερ που δουλεύουν για την εταιρία» αντιστοιχεί σε κανόνα αφού μέσω αυτής ορίζονται τα κριτήρια που πρέπει να πληρούνται για να χαρακτηριστεί κάποιος διευθυντικό στέλεχος της εταιρίας.

Τα γεγονότα και οι κανόνες είναι προτάσεις Horn που έχουν την εξής γενική μορφή:

$$headPr(l_h) \leftarrow bodyPr_1(l_1), \dots, bodyPr_n(l_n). \qquad n \geq 0$$

Το αριστερό μέρος αποτελεί την κεφαλή και το δεξί το σώμα. Τα δομικά στοιχεία των προτάσεων αυτών είναι τα άτομα. Κάθε άτομο περιλαμβάνει ένα κατηγόρημα[25] και μία λίστα όρων $(l)$ με μήκος ίσο με την τάξη[26] του κατηγορήματος. Κάθε όρος μπορεί να είναι είτε μεταβλητή είτε σταθερά. Στη συνέχεια οι μεταβλητές θα συμβολίζονται με κεφαλαίο το πρώτο γράμμα του ονόματός τους ενώ οι σταθερές με μικρό. Στην κεφαλή κάθε πρότασης που αποτελεί μέρος ενός προγράμματος datalog βρίσκεται ακριβώς ένα άτομο. Η διαφορά μεταξύ των κανόνων και των γεγονότων βρίσκεται στο σώμα. Οι κανόνες έχουν τουλάχιστον ένα άτομο στο σώμα ενώ στα γεγονότα το σώμα είναι κενό.

Ένα παράδειγμα κανόνα datalog είναι ο παρακάτω, που αντιστοιχεί στην πρόταση που ορίζει τα διευθυντικά στελέχη της εταιρίας com, είναι: $comExecutive(X) \leftarrow manager(X)$, $worksFor(X, com)$. Το άτομο $worksFor(X, com)$ αναφέρεται στο κατηγόρημα $worksFor$ και σαν όρους περιλαμβάνει την μεταβλητή $X$ και τη σταθερά $com$. Η εμφάνιση του άτομου αυτού στο σώμα ενός κανόνα έχει ως συνέπεια να περιορίζονται οι τιμές της μεταβλητής $X$ στους υπαλλήλους της εταιρίας $com$. Όμοια το πρώτο άτομο στο σώμα του κανόνα περιορίζει τις τιμές της μεταβλητής στο σύνολο όλων των διευθυντών. Το σώμα των κανόνων datalog συμβολίζει την σύζευξη των περιορισμών που προκύπτουν από τα άτομα που περιέχει. Δηλαδή αν κάποια ανάθεση τιμών στις μεταβλητές του κανόνα έχει ως αποτέλεσμα όλα τα άτομα του σώματος να παίρνουν αληθή τιμή τότε οι ίδιες τιμές θα επαληθεύουν και το άτομο στην κεφαλή. Αν εξυπηρετεί η διάζευξη κάποιων συνόλων περιορισμών τότε

[25] Σε αυτό το σημείο πρέπει να σημειωθεί ότι πολύ συχνά στη βιβλιογραφία, ειδικά σε κείμενα που αναφέρονται σε βάσεις δεδομένων, αντί του όρου «κατηγόρημα», ο οποίος έχει τις ρίζες του στο πεδίο της λογικής, χρησιμοποιείται ο όρος «σχέση».

[26] Τάξη (arity): Ο αριθμός των όρων ενός κατηγορήματος.

απαιτείται να κωδικοποιηθεί κάθε σύνολο στο σώμα ενός διαφορετικού κανόνα με το ίδιο κατηγόρημα στο άτομο της κεφαλής. Με αυτό τον τρόπο διασφαλίζεται ότι τα γεγονότα που θα παραχθούν για το κατηγόρημα της κεφαλής πληρούν όλους τους διαφορετικούς συνδυασμούς των απαραίτητων προϋποθέσεων. Ο ισχυρισμός «ο nick δουλεύει στην εταιρία com» κωδικοποιείται στο γεγονός $worksFor(nick, com) \leftarrow$ . Το σώμα είναι κενό αφού το γεγονός ισχύει χωρίς να απαιτείται να συντρέχουν κάποιες προϋποθέσεις.

Για να διατηρηθούν τα επιθυμητά χαρακτηριστικά της datalog σε σχέση με την πλήρη προτασιακή λογική πρώτης τάξης οι κανόνες και τα γεγονότα που συνθέτουν τα προγράμματα datalog οφείλουν να υπακούν σε έναν ακόμη περιορισμό, τη λεγόμενη συνθήκη ασφαλείας. Κάθε μεταβλητή που περιλαμβάνεται στο άτομο της κεφαλής πρέπει να εμφανίζεται και σε τουλάχιστον ένα από τα άτομα του σώματος. Αν έχουμε κάποια μεταβλητή στην κεφαλή η οποία δεν εμφανίζεται στο σώμα τότε δεν υπάρχει κάποιος περιορισμός στις τιμές που μπορεί να λάβει. Αν το σώμα είναι ικανοποιήσιμο τότε ως αποτέλεσμα θα έχουμε την δημιουργία απεριόριστου αριθμού γεγονότων όπου η μεταβλητή θα παίρνει διαφορετικές τιμές [29]. Ο περιορισμός αυτός εξασφαλίζει ότι τα προγράμματα datalog θα έχουν πάντα πεπερασμένη έξοδο. Η συνθήκη ασφαλείας έχει ως συνέπεια να μην είναι δυνατό να περιέχονται μεταβλητές στα γεγονότα.

Το σύνολο των κατηγορημάτων που εμφανίζονται στο πρόγραμμα datalog χωρίζονται σε δύο κατηγορίες ανάλογα με το αν μπορούν να λάβουν νέα πληροφορία κατά την εκτέλεση των κανόνων. Τα κατηγορήματα που εμφανίζονται σε κεφαλή κανόνα μπορούν να εμπλουτιστούν με νέα πληροφορία και αποκαλούνται κατηγορήματα IDB (Intentional Data Base). Τα κατηγορήματα IDB αποτελούν μια υπονοούμενη βάση δεδομένων που ορίζεται από τους κανόνες του προγράμματος και θα προκύψει μετά την πλήρη αποτίμηση του. Αντιθέτως αυτά που απαντώνται είτε σε σώματα κανόνων είτε σε γεγονότα παραμένουν σταθερά και ονομάζονται EDB κατηγορήματα (Extensional Data Base). Η αρχική πληροφορία που υπήρχε για αυτά, είτε σε κάποια εξωτερική βάση δεδομένων είτε με τη μορφή γεγονότων μέσα στον κώδικα του προγράμματος, δεν είναι δυνατό να μεταβληθεί με την αποτίμηση των κανόνων.

Πλέον έχουμε έναν ακόμα τρόπο να ορίσουμε τα συζευκτικά ερωτήματα (CQ). Ένα ερώτημα Q σε datalog ορίζεται από ένα σύνολο $\{Q_\kappa, Q_\pi\}$, όπου $Q_\kappa$ είναι το ειδικό κατηγόρημα που χρησιμοποιείται για την ανάκτηση της απάντησης και $Q_\pi$ είναι ένα πρόγραμμα datalog. Το ερώτημα Q είναι μια ένωση συζευκτικών ερωτημάτων (UCQ) εάν το $Q_\kappa$ είναι το μόνο IDB κατηγόρημα στο $Q_\pi$ και το $Q_\kappa$ δεν βρίσκεται στο σώμα κανενός κανόνα του $Q_\pi$. Ενώ αν το Q είναι ένα UCQ και το $Q_\pi$ περιέχει μόνο έναν κανόνα τότε το Q είναι ένα CQ. Μία πλειάδα σταθερών $\vec{\pi}$ αποτελεί μια απάντηση στο ερώτημα Q πάνω σε μία βάση γνώσης $K = \{\mathcal{T}, \mathcal{A}\}$ αν και μόνο αν $K \cup Q_\pi \vDash Q_\kappa(\vec{\pi})$.

## *3.2 Σημασιολογία*

Πριν αναφερθούμε στην αποτίμηση της μεταγραφής σε datalog πρέπει να εξετάσουμε τη σημασιολογία ενός προγράμματος datalog. Τι περιγράφει αυτό το σύνολο από κανόνες και τα γεγονότα που το συνοδεύουν; Υπάρχουν τρείς διαφορετικοί, αλλά ισοδύναμοι, τρόποι να του δώσουμε νόημα. Όπως θα δούμε η σημασία των προγραμμάτων datalog μπορεί να καθοριστεί με καθαρά διαδικαστικό τρόπο. Τέλος η δηλωτική φύση της γλώσσας υπογραμμίζεται από το γεγονός ότι σε καμία από τις παρακάτω μεθόδους δεν έχει επίδραση στο τελικό αποτέλεσμα η σειρά με την οποία γράφονται οι προτάσεις του προγράμματος ή τα άτομα στο σώμα των κανόνων.

### *3.2.1 Θεωρία μοντέλου*

Ένας τρόπος να δώσουμε νόημα στα προγράμματα datalog είναι μέσω της θεωρίας μοντέλου (model theory) [96]. Η datalog αντιστοιχεί σε ένα υποσύνολο της λογικής πρώτης τάξης και μπορεί να αποδοθεί χωρίς να απαιτείται βαθιά γνώση της θεωρίας. Θα ξεκινήσουμε από τις σταθερές. Στο πεδίο της λογικής πρώτης τάξης είναι δυνατό να τις ερμηνεύσουμε αντιστοιχίζοντάς τες σε οποιοδήποτε στοιχείο του κόσμου που μοντελοποιούμε. Για παράδειγμα οι σταθερές $nick$ και $com$ που εμφανίζονται στο γεγονός $worksFor(nick, com) \leftarrow$ μπορούν να αναπαριστούν το ίδιο αντικείμενο! Για την ερμηνεία των λογικών προτάσεων που αντιστοιχούν σε προγράμματα datalog απαιτείται η χρήση της ταυτοτικής συνάρτησης για να αντιστοιχιστεί κάθε αντικείμενο με μία και μοναδική σταθερά. Οι ερμηνείες αυτού του τύπου αποκαλούνται ερμηνείες Herbrand. Δύο διαφορετικές ερμηνείες Herbrand μπορούν να διαφοροποιούνται μόνο στην ερμηνεία των κατηγορημάτων. Για παράδειγμα το γεγονός $worksFor(nick, com) \leftarrow$ σε κάποια ερμηνεία μπορεί να είναι αληθές ενώ σε άλλη όχι. Το σύνολο όλων των γεγονότων που μπορούν να εμπλακούν στην αποτίμηση ενός προγράμματος datalog P αποκαλείται βάση Herbrand του P. Αποτελείται από όλα τα άτομα που μπορούν να κατασκευαστούν χρησιμοποιώντας τα κατηγορήματα που εμφανίζονται στο P και αντικαθιστώντας τους όρους τους με σταθερές. Μία ερμηνεία Herbrand μπορεί να οριστεί από το σύνολο των γεγονότων που αληθεύουν στα πλαίσια της συγκεκριμένης ερμηνείας άρα αποτελεί ένα υποσύνολο της βάσης Herbrand. Ένα γεγονός αληθεύει με βάση την ερμηνεία $I$ αν περιέχεται σε αυτήν. Ένας κανόνας $A_h \leftarrow A_1, \dots, A_n$ ικανοποιείται από την ερμηνεία $I$ αν για κάθε αντιστοίχηση $\theta$ των μεταβλητών του σε

σταθερές η οποία έχει ως αποτέλεσμα να επαληθεύεται το σώμα, $A_i\theta \in I$ για $1 \leq i \leq n$, ισχύει επίσης $A_h\theta \in I$. Μια ερμηνεία που ικανοποιεί το σύνολο του προγράμματος P αποκαλείται μοντέλο (Herbrand) του P.

Οι κανόνες datalog μπορούν να αντιστοιχιστούν σε προτάσεις λογικής πρώτης τάξης, όπως φαίνεται παρακάτω.

Ο κανόνας:

$$headPr(l_h) \leftarrow bodyPr_1(l_1), \dots, bodyPr_n(l_n).$$

Αντιστοιχεί στην πρόταση[27]:

$$\forall x_{h1}, \dots, x_{hm}\big(headPr(l_h) \leftarrow bodyPr_1(l_1), \dots, bodyPr_n(l_n)\big) \qquad x_{h1}, \dots, x_{hm} \in l_h$$

Η οποία με τη σειρά της γράφεται:

$$\forall x_{h1}, \dots, x_{hm}\left(\exists x_{b1}, \dots, x_{bk}\big(bodyPr_1(l_1), \dots, bodyPr_n(l_n) \rightarrow headPr(l_h)\big)\right)$$
$$x_{h1}, \dots, x_{hm} \in l_h\,, \qquad x_{b1}, \dots, x_{bk} \in l_1 \cup \dots \cup\ l_n$$

Ισοδύναμα μπορούμε να την γράψουμε στην μορφή:

$$\forall x_{h1}, \dots, x_{hm}, x_{b1}, \dots, x_{bk}\big(headPr(l_h) \vee \neg bodyPr_1(l_1) \vee \dots \vee \neg bodyPr_n(l_n)\big)$$
$$x_{h1}, \dots, x_{hm} \in l_h\,, \qquad x_{b1}, \dots, x_{bk} \in l_1 \cup \dots \cup\ l_n$$

Οι προτάσεις αυτού του είδους, που αποτελούνται από τη διάζευξη ενός συνόλου ατόμων με το πολύ ένα από αυτά να εμφανίζεται χωρίς άρνηση, ονομάζονται προτάσεις Horn. Για να ικανοποιείται μια πρόταση Horn, για κάποια ανάθεση τιμών $\theta$, πρέπει είτε να μην αληθεύει κάποιο από τα άτομα $bodyPr_j\big(l_j\theta\big),\ j \in [0, n]$ είτε να αληθεύει το άτομο $headPr(l_h\theta)$. Η αντιστοιχία των προτάσεων Horn με τους κανόνες datalog είναι πλέον προφανής. Αν έστω και ένα άτομο του σώματος δεν αληθεύει τότε ο κανόνας δεν εφαρμόζεται. Αντίθετα αν το σώμα στο σύνολό του είναι αληθές τότε επιβάλλεται να αληθεύει και το άτομο της κεφαλής για να ικανοποιείται ο κανόνας. Τα γεγονότα αντιστοιχούν σε προτάσεις Horn με ένα μοναδικό άτομο το οποίο εμφανίζεται χωρίς άρνηση και λόγω της συνθήκης ασφαλείας δεν δύναται να περιέχει μεταβλητές.

Μπορούμε λοιπόν να θεωρήσουμε το πρόγραμμα datalog ως ένα σύνολο προτάσεων Horn που περιγράφουν τις ιδιότητες που πρέπει να έχουν τα αντικείμενα που αποτελούν τις συμβατές απαντήσεις. Ο πληθυντικός χρησιμοποιείται γιατί κατά κανόνα οι απαντήσεις που επαληθεύουν το σύνολο των προτάσεων είναι άπειρες. Η κάθε μια τους αποτελεί ένα μοντέλο του κόσμου στο οποίο όλες οι προτάσεις του προγράμματος είναι αληθείς. Αν σε ένα ήδη

[27] Το $\leftarrow$ είναι το σύμβολο της λογικής συνεπαγωγής.

υπάρχον μοντέλο προστεθεί οποιαδήποτε πληροφορία που δεν αντιβαίνει σε κάποιον από τους κανόνες τότε καταλήγουμε με μία ακόμα συμβατή απάντηση, διαφορετική από την προηγούμενη, η οποία όμως συνεχίζει να διατηρεί όλες τις περιγραφόμενες από τους κανόνες ιδιότητες. Όπως γίνεται φανερό υπάρχει μεγάλη ασάφεια ως προς τη σημασιολογία του προγράμματος datalog αν ο ορισμός μείνει σε αυτό το σημείο. Πρέπει να επιλέξουμε ποιο από όλα αυτά τα μοντέλα θα ορίσουμε ως τη σημασία αυτών των προτάσεων, ποιο θα δεχθούμε ότι περιγράφουν. Αποδεικνύεται ότι σε κάθε περίπτωση υπάρχει ένα ελάχιστο μοντέλο το οποίο είναι και μοναδικό [97]. Ελάχιστο υπό την έννοια ότι περιέχει τα λιγότερα γεγονότα που απαιτούνται για να ικανοποιηθούν οι προτάσεις του προγράμματος. Ισοδύναμα το ελάχιστο μοντέλο μπορεί να οριστεί ως η τομή όλων των δυνατών μοντέλων. Σύμφωνα με την ιδιότητα τομής μοντέλων (model intersection property) η τομή ενός, πιθανά άπειρου, αριθμού μοντέλων Herbrand ενός συνόλου προτάσεων παραμένει μοντέλο Herbrand των προτάσεων αυτών [98]. Η ιδιότητα αυτή γίνεται εύκολα αντιληπτή αν εφαρμόσουμε τα πορίσματα της θεωρίας συνόλων στις διάφορες ερμηνείες εκφρασμένες με τη μορφή υποσυνόλων της βάσης Herbrand. Με άλλα λόγια η επιλογή του ελάχιστου μοντέλου συνεπάγεται ότι θεωρούμε αληθή μόνο τα γεγονότα που πρέπει να ισχύουν σε όλα τα μοντέλα που μπορούν να ικανοποιήσουν τις προτάσεις του προγράμματος.

### *3.2.2 Θεωρία αποδείξεων*

Για την απόδοση νοήματος μπορούμε επίσης να αποταθούμε στην θεωρία αποδείξεων. Πρόκειται για ένα παρακλάδι της μαθηματικής λογικής που θεμελιώθηκε από τον Hilbert [99]. Αξιοποιώντας τις ιδιότητες της λογικής συνεπαγωγής μπορούμε να δημιουργήσουμε μια διαδικασία απόδειξης της ισχύος των διάφορων γεγονότων. Ένα γεγονός είναι μέρος της απάντησης εάν είναι δυνατό να αποδειχθεί αληθές μέσω της διαδοχικής εφαρμογής κανόνων του προγράμματος με την αξιοποίηση των αρχικών, δεδομένων, γεγονότων. Μπορούμε να επιτύχουμε την απόδειξη με δύο τρόπους. Ένας είναι έχοντας ως αφετηρία τα αρχικά γνωστά γεγονότα να προχωρήσουμε στην εφαρμογή των κανόνων σε αυτά δημιουργώντας όλα τα νέα γεγονότα που είναι δυνατό να αποδειχθούν. Αυτός ο τρόπος αποτελεί την από κάτω προς τα πάνω προσέγγιση. Αν ενδιαφέρει η απόδειξη ενός συγκεκριμένου γεγονότος δεν είναι απαραίτητο να αποδειχθούν όλα τα γεγονότα που υπονοούνται από το πρόγραμμα. Η απόδειξη μπορεί να προσεγγιστεί αντίστροφα, από πάνω

προς τα κάτω[28]. Ξεκινώντας από το προς απόδειξη γεγονός εφαρμόζουμε αντίστροφα τους κανόνες με στόχο να καταλήξουμε σε ένα υποσύνολο των αρχικών γεγονότων. Αυτού του είδους η στρατηγική βασίζεται στην αρχή της ανάλυσης του Robinson [100] και πιο συγκεκριμένα στην παραλλαγή SLD [101], [102]. Η διαδικασία εκκινεί με μία πρόταση στόχο G (goal clause), η οποία είναι η σύζευξη των ατόμων που περιγράφουν τα προς απόδειξη γεγονότα. Οι κανόνες datalog αξιοποιούνται μέσω της συνάρτησης αντικατάστασης θ. Η συνάρτηση θ αντιστοιχεί τις μεταβλητές που περιλαμβάνει ο κανόνας σε όρους. Θα συμβολίζουμε την έκφραση που προκύπτει από την αντικατάσταση των μεταβλητών της Ε σύμφωνα με την θ με Εθ. Με την ανάλυση SLD προσπαθούμε να εντοπίσουμε μία αντικατάσταση θ τέτοια ώστε η έκφραση Gθ να έπεται λογικά από το δοθέν πρόγραμμα datalog P. Σε κάθε βήμα επιλέγεται ένα άτομο από τον στόχο και αναζητείται η πιο γενική αντικατάσταση θ που θα οδηγήσει στην ενοποίηση με κάποιον κανόνα ή γεγονός του P. Έστω ότι ο στόχος G είναι $\leftarrow A_1, \dots, A_n$, ο $H \leftarrow B_1, \dots, B_m$ είναι ένας κανόνας του προγράμματος και επιλέγεται το άτομο $A_i$ για το επόμενο βήμα. Αν βρεθεί αντικατάσταση $\theta$ τέτοια ώστε $A_i\theta = H\theta$ ο στόχος γίνεται $\leftarrow (A_1, \dots, A_{i-1}, B_1, \dots, B_m, A_{n+1}, \dots, A_n)\theta$. Αν η ενοποίηση επιτυγχανόταν με το γεγονός $H \leftarrow$ τα άτομα του στόχου θα ελαττώνονταν κατά ένα και ο νέος στόχος θα ήταν $\leftarrow (A_1, \dots, A_{i-1}, A_{n+1}, \dots, A_n)\theta$. Ο τρόπος αλληλεπίδρασης των στόχων με τους κανόνες και τα γεγονότα γίνεται καλύτερα αντιληπτός μέσω της αναπαράστασης των στοιχείων αυτών με προτάσεις Horn. Μετά την εφαρμογή της αντικατάστασης θ ο στόχος G είναι: $\neg A_1\theta \vee \dots \vee \neg A_i\theta \vee \dots \vee \neg A_n\theta$ ενώ ο κανόνας γράφεται: $H\theta \vee \neg B_1\theta \vee \dots \vee \neg B_m\theta$. Επειδή $A_i\theta = H\theta(1)$ οι δύο προτάσεις έχουν ένα κοινό άτομο το οποίο στη μία εμφανίζεται με το σύμβολο της άρνησης ενώ στην άλλη όχι. Ενώνοντας τις δύο προτάσεις δημιουργείται το $H\theta \vee \neg A_i\theta$ το οποίο λόγω της (1) γίνεται $H\theta \vee \neg H\theta$ που αποτελεί ταυτότητα και μπορεί να παραληφθεί από την τελική έκφραση $(\neg A_1 \vee \dots \vee \neg A_{i-1} \vee \neg B_1 \vee \dots \vee \neg B_m \vee \neg A_{n+1} \vee \dots \vee \neg A_n)\theta$. Ο σκοπός είναι να καταλήξουμε στον κενό στόχο. Αν αυτό επιτευχθεί η έκφραση που κωδικοποιήθηκε στον αρχικό στόχο υπονοείται από τους κανόνες με βάση τα αρχικά δεδομένα. Η όλη διαδικασία μπορεί να παρασταθεί με ένα δένδρο με ρίζα τον στόχο G. Σε κάθε ακμή σημειώνεται η αντικατάσταση θ και ο κανόνας ή το γεγονός με το οποίο έγινε η ενοποίηση. Καταλήγουμε λοιπόν με μία δομή μέσω της οποίας μπορούμε να δικαιολογήσουμε το πώς φτάσαμε σε κάθε νέο συμπέρασμα. Η διαδικασία είναι μη ντετερμινιστική καθώς είναι τυχαία η επιλογή του ατόμου του στόχου και της πρότασης datalog με την οποία θα ενοποιηθεί. Οι επιλογές αυτές παίζουν κρίσιμο ρόλο αφού το ίδιο δένδρο SLD μπορεί να περιέχει κλάδο που δεν τερματίζει,

[28] Οι όροι από κάτω προς τα πάνω (bottom-up) και από πάνω προς τα κάτω (top-down) στην βιβλιογραφία εναλλακτικά αποκαλούνται forward-chaining και backward chaining αντίστοιχα.

κλάδο που μπλοκάρει χωρίς να είναι δυνατή καμία περεταίρω ενοποίηση και κλάδο που να οδηγεί στον κενό στόχο. Ο στόχος αποδεικνύεται αν έστω και ένας κλάδος καταλήγει στον κενό στόχο. Περισσότερες λεπτομέρειες για την ανάλυση SLD μπορούν να αναζητηθούν στα [94], [97], με το τελευταίο να αναφέρεται σε SLD ανάλυση σε προτάσεις Horn με συναρτησιακά σύμβολα.

Η πιθανότητα ύπαρξης ατέρμονων κλάδων στα δένδρα SLD περιπλέκει την από πάνω προς τα κάτω αποτίμηση των προγραμμάτων datalog. Προσπαθώντας από πάνω προς τα κάτω να αποδείξουμε την ισχύ κάποιου γεγονότος μπορεί να βρεθούμε σε ατέρμονα κλάδο ενώ η από κάτω προς τα πάνω προσέγγιση θα μπορούσε να δώσει απάντηση σε πεπερασμένο χρόνο [103]. Το πρόβλημα μπορεί να ξεπεραστεί αν χρησιμοποιήσουμε τεχνικές διάσχισης πρώτα κατά πλάτος (DFS) όπως ο αλγόριθμος query-subquery (QSQ) [104] που τερματίζουν πάντα [105]. Τον αλγόριθμο αυτόν εξομοιώνει η τεχνική της μαγικής μεταγραφής που παρουσιάζεται σε επόμενο κεφάλαιο.

### *3.2.3 Σταθερό σημείο*

Εναλλακτικά μπορούμε να ερμηνεύσουμε το πρόγραμμα datalog με την σημασιολογία σταθερού σημείου (fixpoint semantics). Για αυτό θα ορίσουμε τον τελεστή άμεσης συνέπειας T. Ο τελεστής αυτός λαμβάνει ως ορίσματα ένα σύνολο κανόνων datalog και ένα στιγμιότυπο της βάσης με τα αρχικά γεγονότα. Σαν αποτέλεσμα επιστρέφει το αρχικό αυτό στιγμιότυπο επαυξημένο με όλα τα γεγονότα που μπορούν να εξαχθούν με απλή εφαρμογή των κανόνων στα γεγονότα της βάσης. Γίνεται άμεσα αντιληπτό ότι ο τελεστής αυτός είναι μονότονος αφού διατηρούνται τα αρχικά γεγονότα και επιπλέον δεν υπάρχει κάποιος τρόπος να διαγραφούν δεδομένα από κάποιο κανόνα. Με διαδοχικές εφαρμογές του τελεστή άμεσης συνέπειας «χτίζουμε» από κάτω προς τα πάνω την απάντηση του προγράμματος datalog. Η διαδικασία αυτή του χτισίματος τελειώνει όταν κάποια εφαρμογή του τελεστή δώσει τα ίδια αποτελέσματα με την προηγούμενη. Από αυτό το σημείο και μετά ο τελεστής δεν παράγει νέα αποτελέσματα, διότι όποιο γεγονός μπορούσε να αποδειχθεί αληθές από τα αρχικά γεγονότα και τους κανόνες έχει ήδη αποδειχθεί ή ισοδύναμα όποια πληροφορία μπορούσε να εξαχθεί με τους κανόνες από τα αρχικά δεδομένα έχει εξαχθεί. Έχουμε πλέον φτάσει στο σταθερό σημείο. Τα γεγονότα που αποτελούν το αποτέλεσμα της τελευταίας εφαρμογής του τελεστή άμεστης συνέπειας συνθέτουν ένα μοντέλο που ικανοποιεί τους κανόνες του προγράμματος datalog ενώ παράλληλα περιλαμβάνει τα γεγονότα που αποτελούσαν την αρχική πληροφορία.

Το σύνολο των γεγονότων από τα οποία μπορεί να απαρτίζεται η απάντηση σε ένα πρόγραμμα datalog ή ισοδύναμα το σύνολο των γεγονότων τα οποία οφείλουμε να ελέγξουμε αν περιέχονται στο ελάχιστο μοντέλο που αντιστοιχεί στο πρόγραμμα, είναι πεπερασμένο. Αποτελείται από δύο κατηγορίες γεγονότων. Από τα αρχικά γεγονότα που υπάρχουν στη βάση, τα οποία περιλαμβάνονται στο σύνολό τους στο μοντέλο, και από τα γεγονότα που μπορούν να δημιουργηθούν λαμβάνοντας υπόψη όλους τους δυνατούς συνδυασμούς των σταθερών που συναντάμε στο πρόγραμμα και στα αρχικά δεδομένα με τα κατηγορήματα που βρίσκονται σε κεφαλή κάποιου κανόνα (IDB). Περιορίζουμε την αναζήτηση σε αυτού του είδους τα κατηγορήματα καθώς είναι τα μόνα που μπορούν να δεχθούν νέα πληροφορία. Το γεγονός ότι η αρχική πληροφορία που βρίσκεται στην βάση δεδομένων είναι πεπερασμένη, όπως και ο αριθμός των προτάσεων του προγράμματος, σε συνδυασμό με τον περιορισμό ασφαλείας που ισχύει για κάθε πρόταση που αποτελεί κανόνα ή γεγονός datalog εξασφαλίζει ότι το σύνολο των γεγονότων που μπορεί να περιληφθούν στην απάντηση είναι επίσης πεπερασμένο. Αυτό συνεπάγεται ότι η διαδικασία υπολογισμού του σταθερού σημείου μέσω του τελεστή άμεσης συνέπειας τερματίζει πάντα. Ο μέγιστος δυνατός αριθμός εφαρμογών του τελεστή μέχρι να ολοκληρωθεί η παραγωγή νέων γεγονότων είναι ίσος με το πλήθος των γεγονότων που μπορούν να μετέχουν στο (ελάχιστο) μοντέλο. Όπως αναφέρθηκε προηγουμένως, η διαδικασία τερματίζει όταν κάποια εφαρμογή του τελεστή δεν οδηγήσει σε παραγωγή νέου γεγονότος. Στην ακραία περίπτωση που παράγεται μόνο ένα γεγονός σε κάθε βήμα θα απαιτηθεί ο μέγιστος αριθμός επαναλήψεων.

Για να αποδείξουμε ότι αυτός ο ορισμός είναι ισοδύναμος με τους προηγούμενους πρέπει να δείξουμε ότι το μοντέλο αυτό είναι και το ελάχιστο. Αρχικά θα δούμε ότι τα γεγονότα που περιλαμβάνει το σταθερό σημείο στο οποίο φτάνουμε με τον τρόπο που περιγράφηκε προηγουμένως είναι τα λιγότερα δυνατά και στη συνέχεια ότι αυτό αποδεικνύεται αρκετό για να βεβαιωθούμε ότι έχουμε το ελάχιστο μοντέλο, όπως ορίστηκε με στην προηγούμενη ενότητα.

Θα συμβολίσουμε το ελάχιστο μοντέλο ή ισοδύναμα την τομή όλων των δυνατών μοντέλων, για ένα datalog πρόγραμμα $P$ και αρχικές πληροφορίες στη βάση δεδομένων $A$ με $P(A)$. Ο τελεστής άμεσης συνέπειας που παράγει τα γεγονότα που αποτελούν άμεση συνέπεια των κανόνων του προγράμματος $P$ συμβολίζεται με $T_P$.

Ας θεωρήσουμε ένα οποιοδήποτε σταθερό σημείο Σ για τον τελεστή $T_P$ με αρχικά γεγονότα Α. Ισχύει ότι $\Sigma \supseteq A = T_P^0\{A\}$, με τον εκθέτη στο $T_P^0$ να δείχνει πόσες φορές εφαρμόστηκε ο τελεστής άμεσης συνέπειας. Εφαρμόζοντας τον $T_P$ και στα δύο μέρη έχουμε: $T_P\{\Sigma\} = \Sigma \supseteq T_P^1\{A\}$. Με επαγωγή στον αριθμό εφαρμογών του $T_P$ παίρνουμε: $\Sigma \supseteq T_P^i\{A\}$ για κάθε i. Έτσι αποδεικνύεται ότι το σταθερό σημείο στο οποίο φτάνουμε με την παραπάνω διαδικασία είναι

το ελάχιστο αφού το αποτέλεσμα της εφαρμογής του τελεστή άμεσης συνέπειας είναι πάντα ίσο ή υποσύνολο κάθε σταθερού σημείου άρα και του ελάχιστου.

Το μοντέλο $P(A)$ αποτελεί σταθερό σημείο για τον τελεστή $T_P$ αφού:

- $T_P\{P(A)\} \subseteq P(A)$ διότι το $P(A)$ όντας μοντέλο του $P$ με είσοδο $A$ περιέχει ήδη όσα γεγονότα μπορούν να αποδειχθούν από τους κανόνες του με αρχικά γεγονότα $A$ οπότε ο $T_P$ δε θα προσθέσει κάποιο νέο γεγονός.
- $P(A) \subseteq T_P\{P(A)\}$ από την μονοτονία του $T_P$ έχουμε $T_P\{P(A)\} \subseteq P(A) \Rightarrow T_P\{T_P\{P(A)\}\} \subseteq T_P\{P(A)\}$. Από την τελευταία συνάγουμε ότι το $T_P\{P(A)\}$ αποτελεί μοντέλο του $P$ που περιέχει το $A$, αφού ισχύει $A \subseteq P(A)$ και ο τελεστής άμεσης συνέπειας δεν παράγει επιπλέον πληροφορία. Επειδή όμως το $P(A)$ είναι το ελάχιστο μοντέλο του $P$ που περιέχει το $A$ ισχύει η $P(A) \subseteq T_P\{P(A)\}$.

Όμως κάθε σταθερό σημείο του $T_P$ που περιέχει το $A$ αποτελεί μοντέλο του $P$ για είσοδο $A$, άρα περιέχει το ελάχιστο μοντέλο $P(A)$. Επομένως το $P(A)$ αποτελεί και το ελάχιστο σταθερό σημείο του $T_P$ για είσοδο $A$. Άρα για κάθε πρόγραμμα datalog $P$ με είσοδο $A$ το σταθερό σημείο του τελεστή $T_P$ ταυτίζεται με το ελάχιστο μοντέλο $P(A)$.

## *3.3 Αποτίμηση*

Η διαδικασία υπολογισμού του σταθερού σημείου μας δίνει μία καλή ιδέα για το πώς θα μπορούσε να γίνει η αποτίμηση του προγράμματος datalog. Υπάρχει όμως ένα μεγάλο ελάττωμα. Κάθε φορά που εφαρμόζουμε τον τελεστή άμεσης συνέπειας υπολογίζονται ξανά τα αποτελέσματα όλων των προηγούμενων βημάτων. Αυτό συμβαίνει διότι τα γεγονότα που αποτελούσαν το στιγμιότυπο της βάσης στο προηγούμενο βήμα θα περιληφθούν, μαζί με τα νέα που παρήχθησαν με την εφαρμογή των κανόνων πάνω σε αυτά, στο νέο στιγμιότυπο στο οποίο θα εφαρμοστεί εκ νέου ο τελεστής άμεσης συνέπειας. Έτσι θα εφαρμοστούν πάλι οι ίδιοι κανόνες στα ίδια δεδομένα οδηγώντας στον επανυπολογισμό του ίδιου συνόλου γεγονότων.

Η τελευταία γραμμή της προηγούμενης παραγράφου συνοψίζει το πρόβλημα. Όταν κάποιος κανόνας εφαρμοστεί αποκλειστικά σε παλιά γεγονότα, που ήταν ήδη διαθέσιμα στο προηγούμενο βήμα, τότε θα παραχθεί ήδη γνωστό αποτέλεσμα. Αυτή η απλή παρατήρηση μπορεί να κάνει πολύ πιο αποδοτική την αποτίμηση του προγράμματος datalog. Πρέπει λοιπόν να απαιτήσουμε να χρησιμοποιούνται τα νέα γεγονότα που παρήγαγε ο αλγόριθμος

στο προηγούμενο βήμα. Για κανόνες με ένα IDB κατηγόρημα τα πράγματα είναι απλά: εξετάζουμε μόνο τα νέα γεγονότα. Αν όμως έχουμε περισσότερα του ενός IDB κατηγορήματα ο περιορισμός αυτός θα οδηγήσει σε απώλεια της πληρότητας, αφού θα αποκλείσουμε συνδυασμούς παλαιών και νέων γεγονότων που είναι πολύ πιθανό να ενοποιούνται και να δίνουν νέα πληροφορία. Η λύση είναι σε κάθε εφαρμογή κανόνα να χρησιμοποιούνται σε κάποιο από τα IDB κατηγορήματα μόνο τα νέα γεγονότα. Έτσι εξασφαλίζουμε ότι δεν θα εκτελεστεί κάποιος κανόνας πάνω στα ίδια δεδομένα περισσότερες από μία φορές. Αυτό που είναι πέρα από τις δυνατότητες της μεθόδου είναι να αποκλειστεί η πιθανότητα να παραχθεί ξανά γεγονός που είναι ήδη γνωστό αφού η ενοποίηση διαφορετικών συνόλων γεγονότων μπορεί να οδηγήσει στο ίδιο αποτέλεσμα.

Ας δούμε πως μπορούμε να εφαρμόσουμε την παραπάνω ιδέα. Καταρχάς θα ορίσουμε τα σύμβολα $T_i$ και $S_i$ για να αναφερόμαστε στα νέα δεδομένα που παρήχθησαν στο βήμα *i* και στο σύνολο των δεδομένων που έχουν εξαχθεί μέχρι και το βήμα *i* αντίστοιχα. Δηλαδή ισχύει η σχέση $S_i = T_1 \cup T_2 \cup \ldots \cup T_i$. Έστω ότι έχουμε τον κανόνα: $PrA(X) \leftarrow PrB(X,Y)$. Σύμφωνα με τα προηγούμενα θα τον αντιστοιχίσουμε στον προσωρινό κανόνα: $T_{i+1}\_PrA(X) \leftarrow T_i\_PrB(X,Y)$. Έτσι μόνο τα νέα δεδομένα που υπολογίστηκαν στο βήμα *i* θα εξεταστούν από τον αλγόριθμο στο επόμενο βήμα, αφού στα αποτελέσματα προηγούμενων βημάτων ο κανόνας έχει ήδη εφαρμοστεί. Αν στο σώμα κάποιου κανόνα έχουμε δύο IDB κατηγορήματα τότε θα πρέπει να αντιστοιχίσουμε σε αυτόν δύο προσωρινούς κανόνες για να μπορέσουμε να πάρουμε όλους τους συνδυασμούς δεδομένων που υπονοούνται. Σε κάθε προσωρινό κανόνα θα εξετάσουμε τα νέα δεδομένα για ένα από τα δύο IDB κατηγορήματα. Για παράδειγμα στον κανόνα $PrA(X) \leftarrow PrB(X,Y), PrC(Y,Z)$ αντιστοιχούν οι προσωρινοί κανόνες: $T_{i+1}\_PrA(X) \leftarrow T_i\_PrB(X,Y), S_i\_PrC(Y,Z)$ και $T_{i+1}\_PrA(X) \leftarrow S_i\_PrB(X,Y), T_i\_PrC(Y,Z)$. Πιο αναλυτικά οι συνδυασμοί που ελέγχονται είναι οι εξής:

**Πίνακας 3: Συνδυασμοί που ελέγχονται από τον αρχικό τύπο προσωρινών κανόνων.**

| Προσωρινός κανόνας | Συνδυασμοί που ελέγχονται |
|---|---|
| $T_2\_PrA(X) \leftarrow T_1\_PrB(X,Y), S_1\_PrC(Y,Z)$ | $\mathbf{T_1\_PrB \times T_1\_PrC}$ |
| $T_2\_PrA(X) \leftarrow S_1\_PrB(X,Y), T_1\_PrC(Y,Z)$ | $\mathbf{T_1\_PrB \times T_1\_PrC}$ |
| $T_3\_PrA(X) \leftarrow T_2\_PrB(X,Y), S_2\_PrC(Y,Z)$ | $T_2\_PrB \times T_1\_PrC$, $\mathbf{T_2\_PrB \times T_2\_PrC}$ |
| $T_3\_PrA(X) \leftarrow S_2\_PrB(X,Y), T_2\_PrC(Y,Z)$ | $T_1\_PrB \times T_2\_PrC$, $\mathbf{T_2\_PrB \times T_2\_PrC}$ |
| $T_4\_PrA(X) \leftarrow T_3\_PrB(X,Y), S_3\_PrC(Y,Z)$ | $T_3\_PrB \times T_1\_PrC$, $T_3\_PrB \times T_2\_PrC$, $\mathbf{T_3\_Pr \times T_3\_PrC}$ |
| $T_4\_PrA(X) \leftarrow S_3\_PrB(X,Y), T_3\_PrC(Y,Z)$ | $T_1\_PrB \times T_3\_PrC$, $T_2\_PrB \times T_3\_PrC$, $\mathbf{T_3\_PrB \times T_3\_PrC}$ |

Βλέπουμε ότι γίνεται περιττή δουλειά. Στο βήμα $i$ εκτελείται δύο φορές ο προσωρινός κανόνας με δεδομένα $T_i$*_PrB και* $T_i$*_PrC* αφού το $T_i$ περιέχεται στο $S_i$. Αυτό διορθώνεται πολύ εύκολα αν χρησιμοποιήσουμε σε έναν από τους δύο προσωρινούς κανόνες το $S_{i-1}$ αντί του $S_i$.

**Πίνακας 4: Συνδυασμοί που ελέγχονται από τον βελτιωμένο τύπο προσωρινών κανόνων.**

| Προσωρινός κανόνας | Συνδυασμοί που ελέγχονται |
|---|---|
| $T_2$*_PrA(X)* ← $T_1$*_PrB(X,Y),* $S_0$*_PrC(Y,Z)* | $T_1$*_PrB x Ø* |
| $T_2$*_PrA(X)* ← $S_1$*_PrB(X,Y),* $T_1$*_PrC(Y,Z)* | $T_1$*_PrB x* $T_1$*_PrC* |
| $T_3$*_PrA(X)* ← $T_2$*_PrB(X,Y),* $S_1$*_PrC(Y,Z)* | $T_2$*_PrB x* $T_1$*_PrC* |
| $T_3$*_PrA(X)* ← $S_2$*_PrB(X,Y),* $T_2$*_PrC(Y,Z)* | $T_1$*_PrB x* $T_2$*_PrC,* $T_2$*_PrB x* $T_2$*_PrC* |

Η ιδέα που παρουσιάστηκε στο προηγούμενο παράδειγμα επεκτείνεται εύκολα σε κανόνες με οποιονδήποτε αριθμό IDB κατηγορημάτων. Έστω ότι έχουμε έναν κανόνα με n IDB κατηγορήματα $Pr(X) \leftarrow Pr_1(t_1), Pr_2(t_2), \dots, Pr_n(t_n)$. Θα τον αντιστοιχίσουμε σε n προσωρινούς κανόνες. Στην επανάληψη $i$ για $j \in [1, n]$ έχουμε:

$$T_{i+1}Pr(X) \leftarrow S_iPr_1(t_1), \dots, S_iPr_{j-1}(t_{j-1}), T_iPr_j(t_j), S_{i-1}\mathrm{Pr}_{j+1}(t_{j+1}), \dots, S_{i-1}Pr_n(t_n)$$

Υπάρχει ακόμα ένα ενδιαφέρον σημείο που θα εξηγηθεί καλύτερα μέσω ενός παραδείγματος:

**Παράδειγμα 3: Εκ νέου ανακάλυψη γεγονότος.**

| | |
|---|---|
| Κανόνες: | 1) oneGenBefore (X,Y) ← childOf (Y,X) |
| | 2) grandParentOf (X,Z) ← oneGenBefore (X,Y), childOf (Z,Y) |
| | 3) oneGenBefore (X,Z) ← grandParentOf (X,Y), childOf (Y,Z) |
| Δεδομένα: | childOf (b,a). |
| | childOf (c,b). |

Πρώτα θα εκτελεστεί ο κανόνας 1 επειδή δεν περιέχει κάποιο IDB κατηγόρημα. Δεν χρειάζεται να εκτελεστεί ξανά αφού τα EDB κατηγορήματα παραμένουν σταθερά.

$T_1$_oneGenBefore(X,Y) ← childOf(Y,X)

Δεδομένα μετά την εκτέλεση του κανόνα:

| $T_1$_oneGenBefore | $T_1$_grandParentOf |
|---|---|
| (a,b), (b,c) | Ø |

| $S_1$_oneGenBefore | $S_1$_grandParentOf |
|---|---|
| (a,b), (b,c) | Ø |

- **επανάληψη 1:**

$T_2$_oneGenBefore (X,Z) ← $T_1$_oneGenBefore (X,Y), childOf(Z,Y)

$T_2$_oneGenBefore (X,Z) ← $T_1$_oneGenBefore grandParentOf(X,Y), childOf(Y,Z)

| $T_2$_oneGenBefore | $T_2$_grandParentOf |
|---|---|
| Ø | (a,c) |

| $S_2$_oneGenBefore | $S_2$_grandParentOf |
|---|---|
| (a,b), (b,c) | (a,c) |

- **επανάληψη 2:**

$T_3$_grandParentOf(X,Z) ← $T_2$_oneGenBefore(X,Y), childOf(Z,Y)

$T_3$_oneGenBefore(X,Z) ← $T_2$_grandParentOf(X,Y), childOf(Y,Z)

| $T_3$_oneGenBefore | $T_3$_grandParentOf |
|---|---|
| (a,b) | Ø |

| $S_3$_oneGenBefore | $S_3$_grandParentOf |
|---|---|
| (a,b), (b,c) | (a,c) |

Σε αυτό το σημείο παρατηρούμε ότι το *$T_3$_oneGenBefore* περιέχει το (a,b) το οποίο όμως δεν αποτελεί νέο δεδομένο για το κατηγόρημα *oneGenBefore*. Υπάρχει ήδη στο πρώτο συγκεντρωτικό κατηγόρημα *$S_1$_oneGenBefore* λόγω του *$T_1$_oneGenBefore* στο οποίο αρχικά εμφανίστηκε. Καταλήγουμε λοιπόν στο συμπέρασμα ότι το $T_i$ έτσι όπως έχει οριστεί δεν εξυπηρετεί. Ενώ περιέχει όλα τα δεδομένα που παρήχθησαν στην τελευταία επανάληψη, αυτά πολλές φορές δεν αποτελούν νέα πληροφορία αλλά κάποια μπορεί να έχουν εξαχθεί και κατά συνέπεια ελεγχθεί προηγουμένως. Για να αποφύγουμε και αυτή την περίπτωση περιττών υπολογισμών αντί να χρησιμοποιήσουμε το $T_i$ θα αφαιρέσουμε από αυτό όσα γεγονότα είναι ήδη γνωστά και θα έχουμε $\Delta_i = T_i - S_{i-1}$. Χρησιμοποιώντας στο επόμενο βήμα το $\Delta_i$ διασφαλίζεται ότι θα ληφθούν υπόψη μόνο όσα δεδομένα είναι πραγματικά νέα από αυτά που μόλις υπολογίστηκαν. Δυστυχώς δεν έχουμε τρόπο να αποφύγουμε τον επανυπολογισμό κάποιων γεγονότων αφού οι κανόνες που δίνονται μπορεί να παρέχουν περισσότερες από μία διαδρομές για την εξαγωγή αυτών. Συνοψίζοντας, ο συμβολισμός $T_i$ (από τη λέξη temporary) χρησιμοποιείται για τα προσωρινά νέα δεδομένα, ενώ το πρόθεμα $\Delta_i$ καταδεικνύει δεδομένα που παρήχθησαν για πρώτη φορά στο βήμα $i$. Η τελική μορφή των προσωρινών κανόνων στους οποίους αντιστοιχεί ο κανόνας με n IDB κατηγορήματα $Pr(X) \leftarrow Pr_1(t_1), Pr_2(t_2), \dots, Pr_n(t_n)$ που είδαμε προηγουμένως, λαμβάνοντας υπόψη τον νέο συμβολισμό, για την επανάληψη $i$ και για $j \in [1, n]$ είναι της μορφής:

$$T_{i+1}Pr(X) \leftarrow S_iPr_1(t_1), \dots, S_iPr_{j-1}(t_{j-1}), \Delta_iPr_j(t_j), S_{i-1}\mathrm{Pr}_{j+1}(t_{j+1}), \dots, S_{i-1}Pr_n(t_n)$$

Για τον τερματισμό του αλγόριθμου αρκεί ένας απλός έλεγχος αν έχουν παραχθεί νέα γεγονότα στην τελευταία επανάληψη. Αν έχουν μεν παραχθεί γεγονότα αλλά δεν είναι νέα δεν υπάρχει λόγος να συνεχίσουμε διότι, όπως αναφέρθηκε και στην προηγούμενη ενότητα αυτά έχουν ελεγχθεί ως προς την ενοποίηση στα προηγούμενα βήματα. Αν λοιπόν όλα τα $\Delta_i$ είναι κενά δεν θα παραχθεί τίποτα σε επόμενη επανάληψη αφού όλοι οι κανόνες που εκτελούνται βασίζονται αποκλειστικά στα νέα αποτελέσματα για κάποιο από τα IDB κατηγορήματά τους (περιέχουν υποχρεωτικά ένα κατηγόρημα με πρόθεμα «$\Delta_i$»). Σε αυτό το σημείο δεν υπάρχει κάποιος συνδυασμός γεγονότων που δεν έχει ήδη ελεγχθεί. Τα νέα γεγονότα κάθε βήματος ελέγχονται ως προς την ενοποίηση με όλα όσα έχουν ανακαλυφθεί προηγουμένως και στη συνέχεια ως μη νέα θα ελεγχθούν με όσα θα παραχθούν μετά από αυτά. Έτσι διασφαλίζεται ότι όλοι δυνατοί συνδυασμοί έχουν ελεγχθεί.

Ο αλγόριθμος για την αποτίμηση του προγράμματος datalog στην τελική του μορφή είναι ο εξής:

1) Εκτελούνται όλοι οι κανόνες που δεν περιέχουν IDB κατηγόρημα στο σώμα, με το πρόθεμα «$\Delta_1$» να έχει προστεθεί στο όνομα του κατηγορήματος στην κεφαλή, για να είναι έτοιμα προς αξιοποίηση από τους προσωρινούς κανόνες τα δεδομένα που θα παραχθούν: $\Delta_1\_PrIDB \leftarrow PrEDB1(t_1), \dots, PrEDBn(t_n)$
2) Θέτουμε: i = 1
3) do:
    - για κάθε IDB κατηγόρημα $S$: $S_i = S_{i-1} \cup \Delta_i$
    - για κάθε κανόνα με τουλάχιστον ένα IDB κατηγόρημα στο σώμα:
        - μετάφρασή του στους κατάλληλους προσωρινούς κανόνες:
          $T_{i+1}\_HeadPr \leftarrow S_i\_PrIDB1(t_1), \dots, \Delta_i\_PrIDBj(t_j), \dots, S_{i-1}\_PrIDBm(t_m), \dots, PrEDB1(te_1), \dots, PrEDBn(te_n)$.
        - εκτέλεση των προσωρινών κανόνων
    - για κάθε IDB κατηγόρημα: $\Delta_{i+1} = T_{i+1} - S_i$
    - i = i+1

    while: $\Delta_i \neq \emptyset$ για κάποιο IDB κατηγόρημα
4) για κάθε IDB κατηγόρημα: μετονομασία του τελευταίου ενδιάμεσου αποτελέσματος για την αφαίρεση του προθέματος «$S_{i-1}$» :
   $PrIDB \leftarrow S_{i-1}\_PrIDB$

## *3.4 Τμηματοποίηση του προγράμματος datalog*

Η τεχνική αυτή αναλαμβάνει τη βελτιστοποίηση, σε μεγάλο βαθμό, της απόδοσης της εφαρμογής χωρίζοντας το αρχικό πρόγραμμα datalog σε μικρότερα μέρη τα οποία μπορούν να αποτιμηθούν διαδοχικά και να δώσουν το ίδιο τελικό αποτέλεσμα. Ο περιορισμός της έκτασης της εισόδου του αλγόριθμου αποτίμησης προγραμμάτων datalog οδηγεί σε μεγάλη εξοικονόμηση πόρων και πολύ ταχύτερη εκτέλεση περιορίζοντας δραστικά τον αριθμό των κανόνων που πρέπει να εκτελεστούν για να φτάσουμε στο τελικό αποτέλεσμα. Προτού όμως αναφερθούμε στους λόγους οι οποίοι συντελούν σε αυτό ας δούμε την ιδέα πίσω από αυτή τη βελτίωση. Επειδή εδώ ενδιαφέρει η παραγωγή πληροφορίας, στο υπόλοιπο αυτής ενότητας αναφερόμαστε μόνο σε IDB κατηγορήματα οπότε ο όρος θα παραλείπεται.

Αν εξετάσουμε λεπτομερώς τη διαδικασία εκτέλεσης των προγραμμάτων datalog θα παρατηρήσουμε ότι υπάρχουν στάδια κατά τα οποία ορισμένα μόνο κατηγορήματα μεγαλώνουν σε όγκο πληροφορίας ενώ τα υπόλοιπα παραμένουν στάσιμα, παρόλο που εκτελούνται και για αυτά οι αντίστοιχοι κανόνες. Αυτό μπορεί να συμβαίνει για δύο λόγους: Είτε γιατί δεν υπάρχει ακόμα πληροφορία σε κάποια από τα κατηγορήματα από τα οποία εξαρτώνται, είτε διότι έχουν πλέον υπολογιστεί για αυτά όλες οι πληροφορίες που θα αποτελούν το τελικό αποτέλεσμα και δεν πρόκειται να αυξηθούν άλλο. Αυτό ακριβώς είναι το κριτήριο για τον διαχωρισμό της εισόδου. Να τροφοδοτούμε τον αλγόριθμο μόνο με τους κανόνες που αντιστοιχούν στα κατηγορήματα τα οποία θα «μεγαλώσουν» σε κάθε στάδιο της εκτέλεσης.

Όπως γίνεται φανερό μας ενδιαφέρει η «ροή της πληροφορίας» μεταξύ των κατηγορημάτων, δηλαδή ο τρόπος που εξαρτάται κάθε κατηγόρημα από τα δεδομένα άλλων. Σε αυτό το σημείο θα αναζητήσουμε την συνδρομή της θεωρίας γραφημάτων. Οι εξαρτήσεις αυτές αναδεικνύονται εξαιρετικά μέσω της αναπαράστασης του προγράμματος datalog μέσω ενός κατευθυνόμενου γράφου. Θα συμπεριλάβουμε μία κορυφή για κάθε κατηγόρημα (IDB) του προγράμματος. Οι ακμές που θα αποτυπώνουν κάθε κανόνα στον γράφο θα ξεκινούν από το κατηγόρημα στην κεφαλή και θα καταλήγουν στα κατηγορήματα που περιλαμβάνονται στο σώμα. Μπορεί η φορά των ακμών να είναι αντίθετη με αυτή που διαισθητικά θα περίμενε κανείς να αναπαριστά τη «ροή πληροφορίας», αφού η πληροφορία μεταφέρεται μέσω του κανόνα από τα κατηγορήματα του σώματος σε αυτό στην κεφαλή, αλλά εξυπηρετεί καθαρά τεχνικούς λόγους που θα εξηγηθούν παρακάτω.

Ας ορίσουμε την έννοια του μονοπατιού δεδομένων ως μια διαδρομή στον γράφο που αρχίζει και τελειώνει σε κορυφή και περνά από μια κορυφή στην επόμενη εφ’ όσον ανάμεσά τους υπάρχει ακμή με την κατάλληλη φορά. Το μονοπάτι δεδομένων μπορεί να διέρχεται

περισσότερες από μία φορές από κάποια κορυφή ή κατευθυνόμενο κύκλο στο γράφο πριν καταλήξει στο τέρμα του. Δεν πρέπει να συγχέεται με το μονοπάτι όπως ορίζεται στο μεγαλύτερο μέρος της βιβλιογραφίας της θεωρίας γραφημάτων όπου δεν επιτρέπεται η διέλευση από κάποια κορυφή παραπάνω από μία φορές. Κύκλος στον γράφο προκύπτει από την αμοιβαία αναδρομική εξάρτηση μεταξύ δύο ή περισσότερων κατηγορημάτων. Η πιο απλή μορφή αυτής της εξάρτησης φαίνεται στους κανόνες: *Pr1 ← Pr2, PrC, ..., PrN* και *Pr2 ← Pr1, Prc, ..., Prn*. Το κατηγόρημα *Pr1* εξαρτάται από τα δεδομένα του *Pr2* και το αντίστροφο. Η αλληλεξάρτηση μπορεί να επεκταθεί σε μεγάλο αριθμό κατηγορημάτων και να περιλαμβάνει πολλούς κύκλους. Σε κάθε περίπτωση το κοινό γνώρισμα όλων αυτών των περιπτώσεων είναι ότι ακολουθώντας το μονοπάτι των δεδομένων μπορούμε να βρεθούμε από οποιαδήποτε κορυφή σε οποιαδήποτε άλλη. Οι κορυφές που αντιστοιχούν σε κατηγορήματα με τέτοιου είδους εξάρτηση αποτελούν ισχυρά συνεκτικές συνιστώσες (Ι.Σ.Σ.) του γράφου.

Τώρα πρέπει να δούμε πως θα διαχωρίσουμε τα κατηγορήματα ή ισοδύναμα τις κορυφές. Όπως αναφέρθηκε παραπάνω, ενδιαφέρει να τροφοδοτούμε τον αλγόριθμο κάθε φορά με τους κανόνες για τα κατηγορήματα εκείνα για τα οποία θα έχουμε παραγωγή πληροφορίας. Οπότε αν υπάρχει κάποια Ι.Σ.Σ. στο γράφο, τα κατηγορήματα που την αποτελούν θα πρέπει να αποτιμούνται μαζί, σαν ενιαία ομάδα, γιατί οι πληροφορίες τους θα πληθαίνουν παράλληλα, αφού το ένα θα τροφοδοτεί το άλλο. Αυτό που πρέπει να κάνουμε είναι να υπολογίσουμε τις ισχυρά συνεκτικές συνιστώσες του γράφου και να πάρουμε τις ομάδες των αλληλεξαρτώμενων κατηγορημάτων. Σύμφωνα με αυτές θα ομαδοποιήσουμε τους κανόνες του αρχικού προγράμματος datalog. Όσοι κανόνες έχουν στην κεφαλή κάποιο κατηγόρημα με κορυφή στην ίδια Ι.Σ.Σ. θα μπουν στην και στην ίδια ομάδα. Κανόνες με κεφαλή κατηγόρημα που δεν μετέχει σε κάποιο κύκλο θα εκτελούνται χωριστά. Δηλαδή μια κορυφή που δεν μετέχει σε κάποια Ι.Σ.Σ. θεωρείται από μόνη της Ι.Σ.Σ.

Τέλος μένει να καθοριστεί η σειρά με την οποία θα αποτιμηθούν οι ομάδες κανόνων. Σκοπός είναι όταν έρθει η ώρα της αποτίμησης κάποιας ομάδας να έχουμε έτοιμη την πληροφορία από προηγούμενες που την τροφοδοτούν. Αν δεν ισχύει αυτό είτε θα εκτελούμε τους κανόνες χωρίς να δημιουργούνται νέα δεδομένα είτε θα παράγεται χρήσιμο υπολογιστικό έργο με βάση τη διαθέσιμη πληροφορία αλλά ο υπολογισμός θα πρέπει να επαναληφθεί εκ νέου όταν θα γίνει διαθέσιμη νέα πληροφορία. Εν ολίγοις δεν υπάρχει κάποιο κέρδος από την εκκίνηση της αποτίμησης κάποιας ομάδας πριν την ολοκλήρωση όλων των προηγούμενων. Για να εντοπίσουμε τις προηγούμενες Ι.Σ.Σ. θα ακολουθήσουμε ακμές που ξεκινούν από κάποια κορυφή της τρέχουσας Ι.Σ.Σ. αλλά καταλήγουν εκτός αυτής. Αν συμπυκνώσουμε σε μία κορυφή κάθε Ι.Σ.Σ. θα δημιουργήσουμε ένα νέο κατευθυνόμενο γράφο ο οποίος δεν θα

περιέχει κύκλους. Αυτό που χρειαζόμαστε είναι η (αντίστροφη) τοπολογική διάταξη των κορυφών του νέου γράφου.

Στην θεωρία γραφημάτων υπάρχει ένας αποδοτικός αλγόριθμος για τον υπολογισμό των ισχυρά συνεκτικών συνιστωσών που σαν παραπροϊόν έχει και τον υπολογισμό της τοπολογικής διάταξής τους. Αυτός ο αλγόριθμος παρουσιάστηκε το 1972 από τον Robert E. Tarjan [106]. Ακολουθεί ο αλγόριθμος σε ψευδοκώδικα.

**Εικόνα 1: Ο αλγόριθμος του Tarjan.**

```
index = 0
stack = {}                          //στοίβα με τις κορυφές υπό επεξεργασία, αρχικά κενή
for each v in V do                  //V το σύνολο των κορυφών
        if (v.visited -1) then      //για να διασφαλίσουμε ότι θα διατρέξει όλες τις κορυφές
                Tdfs(v)
        end if
end for

function Tdfs(v)
        v.visited = index           //ο χρόνος (σειρά) στον οποίο επισκεφτήκαμε, για πρώτη φορά,
                                    //  την κορυφή
        v.lowlink = index
        index = index + 1
        stack.push(v)
        v.inStack = true            //για να έχουμε σε σταθερό χρόνο τον έλεγχο αν βρίσκεται στη
                                    //  λίστα
  for each (v, w) in E do           //διατρέχουμε όλες  τις επόμενες κορυφές, Ε το σύνολο των ακμών
   if (w.visited =  -1) then        //  και για κάθε επόμενη κορυφή που δεν έχουμε ακόμα
                                    //  επισκεφτεί τρέχουμε αναδρομικά τον αλγόριθμο
     Tdfs(w)
     v.lowlink  := min(v.lowlink, w.lowlink)
   else if (w.onStack) then         //αν η επόμενη κορυφή είναι ήδη στη λίστα τότε σημαίνει ότι
                                    //  είναι στην ίδια ισχυρά συνεκτική συνιστώσα
     v.lowlink  := min(v.lowlink, w.visited)
   end if
  end for

  if (v.lowlink = v.visited) then //αυτό σημαίνει ότι η v είναι η ρίζα (πρώτη κορυφή που έτυχε να
                                  //  επισκεφτούμε) της ισχυρά συνεκτικής συνιστώσας
    new SCC
    repeat
      w := stack.pop()
      w.inStack:= false
```

```
    SCC = SCC U {w}
  until (w = v)
 end if
end function
```

Το αρχικό for loop είναι απαραίτητο για να διασφαλιστεί ότι ο αλγόριθμος θα εξετάσει όλες τις κορυφές. Αν τύχει να ξεκινήσει η διάσχιση του γράφου από κάποια κορυφή από την οποία δεν υπάρχει κατευθυνόμενο μονοπάτι προς κάποια άλλη τότε ο αλγόριθμος δεν είναι δυνατό να επισκεφτεί τη δεύτερη. Αυτό το χαρακτηριστικό θα το εκμεταλλευτούμε προς όφελός μας κατά την απάντηση συγκεκριμένων ερωτημάτων όπως θα δούμε παρακάτω. Ο αλγόριθμος (συνάρτηση Tdfs) ξεκινά από οποιαδήποτε κορυφή και επισκέπτεται διαδοχικά όλες τις επόμενες αυτής.

Τα πεδία που διατηρούνται για κάθε κορυφή είναι τα:

- **visited:** Δείχνει τη σειρά με την οποία επισκεφθήκαμε, για πρώτη φορά, κάθε κορυφή. Η αρχική του τιμή -1 δείχνει ότι δεν έχουμε εξετάσει ακόμα την κορυφή.
- **lowLink:** Διατηρεί την μικρότερη τιμή visited μεταξύ των κορυφών που ανήκουν στην ίδια ισχυρά συνεκτική συνιστώσα, δηλαδή το visited της κορυφής που έτυχε να επισκεφτούμε πρώτη για την τρέχουσα Ι.Σ.Σ..
- **inStack:** Το πεδίο αυτό χρησιμεύει για τον άμεσο έλεγχο αν κάποια κορυφή είναι στην στοίβα. Αποφεύγουμε έτσι την αναζήτηση μεταξύ των στοιχείων της στοίβας.

Η πρώτη επίσκεψη σε μία κορυφή ξεκινά με την ενημέρωση του πεδίου visited με την τρέχουσα τιμή του index. Την ίδια τιμή παίρνει αρχικά και το lowLink διότι η ίδια η κορυφή θεωρείται από μόνη της μια ισχυρά συνεκτική συνιστώσα. Η τρέχουσα κορυφή προστίθεται στη στοίβα, ενώ ταυτόχρονα ενημερώνεται και το πεδίο inStack και το index αυξάνεται κατά ένα για την επόμενη επίσκεψη σε κορυφή. Στη συνέχεια ο αλγόριθμος θα προχωρήσει στη εξέταση όλων των κορυφών για τις οποίες υπάρχει ακμή από την τρέχουσα.

Αν συναντήσουμε κάποια κορυφή που δεν έχουμε ήδη επισκεφτεί τότε εκτελείται αναδρομικά η Tdfs σε αυτήν, έχουμε δηλαδή διάσχιση πρώτα κατά βάθος (DFS). Μόλις ολοκληρωθεί η εκτέλεση του αλγόριθμου στην επόμενη κορυφή, ενημερώνεται η τιμή του lowLink με την τιμή του ίδιου πεδίου στην επόμενη κρυφή, αν αυτή είναι μικρότερη από την τρέχουσα. Ας σταθούμε λίγο στο τι σημαίνει αυτό. Επειδή το lowLink κρατά την μικρότερη τιμή του visited από όλες τις κορυφές στην ίδια ισχυρά συνεκτική συνιστώσα αν αυτό βρεθεί μικρότερο τότε σημαίνει ότι στην ίδια Ι.Σ.Σ. βρίσκεται κάποια κορυφή που επισκεφτήκαμε πριν την τρέχουσα. Για να είμαστε σε θέση να διακρίνουμε στη συνέχεια την Ι.Σ.Σ. στην

οποία ανήκει η τρέχουσα κορυφή επιβάλλεται να ενημερωθεί η τιμή του, αν προκύψει μικρότερο σε κάποια επόμενη κορυφή.

Για επόμενη κορυφή που έχουμε ήδη επισκεφθεί, ελέγχεται αν βρίσκεται ακόμα στη στοίβα. Αν έχει βγει τότε σημαίνει ότι ανήκει σε διαφορετική Ι.Σ.Σ. οπότε δεν κάνουμε κάτι. Αν είναι ακόμα στη στοίβα τότε επειδή υπάρχει δρόμος από την τρέχουσα κορυφή προς αυτήν αλλά και αντίθετα αφού βρισκόμασταν εκεί πριν έρθουμε στην τρέχουσα, οι δύο κορυφές βρίσκονται στην ίδια Ι.Σ.Σ. οπότε αν το visited της κορυφής αυτής είναι μικρότερο πρέπει να ενημερωθεί η τιμή του lowLink για την τρέχουσα.

Όταν ολοκληρωθεί αυτή η διαδικασία για όλες τις κορυφές για τις οποίες υπάρχει ακμή από την τρέχουσα και πριν επιστρέψει η αναδρομική μέθοδος τον έλεγχο στην μέθοδο που την κάλεσε, εξετάζεται αν το lowLink είναι ίσο με το visited. Αν αυτό ισχύει τότε η τρέχουσα κορυφή είναι η πρώτη που επισκεφτήκαμε από τις κορυφές της συγκεκριμένης Ι.Σ.Σ. πράγμα που σημαίνει ότι όσες κορυφές βρίσκονται πάνω από αυτή στη στοίβα ανήκουν στην ίδια Ι.Σ.Σ. Έτσι αφαιρούμε κορυφές από τη στοίβα μέχρι να καταλήξουμε στην τρέχουσα και τις προσθέτουμε όλες στην ίδια Ι.Σ.Σ.

Ο αλγόριθμος είναι βέλτιστος αφού η Tdfs επισκέπτεται κάθε κορυφή μία φορά, είτε λόγω του αρχικού for είτε μέσω της αναδρομικής κλίσης στο σώμα της συνάρτησης. Το ίδιο ισχύει και για τις ακμές του γράφου. Μέσω του βρόχου for της Tdfs διασφαλίζεται ότι θα εξεταστούν όλες από μία φορά. Τελικά η πολυπλοκότητα του αλγόριθμου είναι O(|V|+|E|), η οποία είναι βέλτιστη αφού είναι απαραίτητο από κάθε αλγόριθμο να εξεταστεί το σύνολο του γράφου τουλάχιστον μία φορά.

### *3.4.1 Κέρδος στην διαδικασία αποτίμησης*

Ο αλγόριθμος αποτίμησης προγραμμάτων datalog που παρουσιάστηκε στο προηγούμενο κεφάλαιο προωθεί σε κάθε επανάληψη του κύριου βρόχου του το σύνολο της διαθέσιμης πληροφορίας ένα βήμα, κατά μήκος δηλαδή μίας ακμής σε όλα τα σημεία όπου υπάρχουν οι κατάλληλες συνθήκες (πληρούνται από τη διαθέσιμη πληροφορία οι περιορισμοί των αντίστοιχων κανόνων). Για να δοθεί το τελικό αποτέλεσμα για κάποιο κατηγόρημα πρέπει να συνδυαστούν οι πληροφορίες από όλα τα μονοπάτια δεδομένων που καταλήγουν σε αυτό. Η ολοκλήρωση της απάντησης θα απαιτήσει τόσες επαναλήψεις όσα είναι και τα βήματα στο μεγαλύτερο μονοπάτι δεδομένων. Αυτός ο αριθμός επηρεάζεται από τα δεδομένα που συνοδεύουν το πρόγραμμα αφού ανάλογα με τις συγκεκριμένες τιμές των μεταβλητών μπορεί να απαιτηθεί να ακολουθηθεί κάποιος κύκλος περισσότερες ή λιγότερες φορές. Δεν

επηρεάζεται όμως από τον διαχωρισμό των κανόνων σε ομάδες, αφού οι κύκλοι περιλαμβάνονται πάντα ολόκληροι στις Ι.Σ.Σ. Η εξοικονόμηση πόρων προέρχεται κατά ένα μέρος από το γεγονός ότι οι κανόνες εκτελούνται μόνο όταν έχουμε διαθέσιμο το σύνολο της πληροφορίας των προηγούμενων ομάδων, δηλαδή μέχρι και ένα βήμα πριν από τα κατηγορήματα της τρέχουσας ομάδας. Έτσι δεν απαιτείται να επαναληφθεί η εκτέλεσή τους για να διέλθει, από το τμήμα του γράφου που αντιστοιχεί στην ομάδα, πληροφορία που δεν έχει παραχθεί όταν εκκίνησε η εκτέλεση την προηγούμενη φορά. Επίσης αποφεύγεται η επιβάρυνση από την συνεχιζόμενη συμμετοχή στη διαδικασία αποτίμησης κανόνων που αντιστοιχούν σε ομάδες για τις οποίες έχει ολοκληρωθεί η εκμαίευση πληροφορίας.

Η επεξεργασία των κανόνων κατά ομάδες έχει ως αποτέλεσμα τον περιορισμό του αριθμού των κανόνων που δίνονται ταυτόχρονα σαν είσοδος στον αλγόριθμο αποτίμησης datalog. Αυτό έχει ως συνέπεια να μειώνεται και ο αριθμός των IDB κατηγορημάτων για τα οποία εξάγουμε πληροφορία παράλληλα. Επιπλέον, κατηγορήματα που μεταχειρίστηκαν ως τέτοια κατά την αποτίμηση προηγούμενων ομάδων κανόνων στις επόμενες θα θεωρούνται EDB αφού η πληροφορία για αυτά έχει εξαχθεί στο σύνολό της και είναι πλέον σταθερή. Ακόμα, λόγω του τρόπου που έγινε ο διαχωρισμός, IDB κατηγορήματα επόμενων ομάδων δεν εμφανίζονται σε κανόνες προηγούμενων. Σε αυτό το σημείο ας θυμηθούμε ότι σε κάθε επανάληψη του κύριου βρόχου του αλγόριθμου, για κάθε κανόνα της εισόδου, πρέπει να εκτελέσουμε τόσους προσωρινούς κανόνες όσα είναι και τα IDB κατηγορήματα που περιέχει στο σώμα του. Περιορίζοντας τον αριθμό αυτών των κατηγορημάτων καταφέρνουμε να μειώσουμε τον αριθμό των προσωρινών κανόνων στους οποίους «μεταφράζονται» οι κανόνες της εισόδου και τελικά να περιορίσουμε σημαντικά τους εκτελεσθέντες κανόνες.

Μια κατηγορία προγραμμάτων datalog που ωφελούνται ιδιαίτερα από τον διαχωρισμό των κανόνων και την τμηματική αποτίμησή τους είναι τα γραμμικά. Τα προγράμματα αυτά παρουσιάζουν την ιδιαιτερότητα να περιλαμβάνουν στα σώματα των κανόνων τους το πολύ ένα κατηγόρημα το οποίο είναι αμοιβαία αναδρομικά εξαρτώμενο από το κατηγόρημα στην κεφαλή του κανόνα. Ο διαχωρισμός των κανόνων τους έχει ως αποτέλεσμα την ανάδειξη αυτής της ιδιαιτερότητας καθώς τα μη εξαρτώμενα κατηγορήματα κάθε κανόνα τοποθετούνται σε προηγούμενες ομάδες. Αυτό έχει ως αποτέλεσμα σε κάθε αρχικό κανόνα να αντιστοιχεί μόνο ένας προσωρινός κανόνας, έτσι μειώνεται δραστικά το κόστος της αποτίμησης του γραμμικού προγράμματος. Ένα παράδειγμα της έκτασης της εξοικονόμησης στον αριθμό των κανόνων βλέπουμε παρακάτω.

**Παράδειγμα 4: Εξοικονόμηση στον αριθμό των εκτελεσθέντων κανόνων μέσω της ομαδοποίησης.**

Κανόνες:

α) $pr1(X,Y) \leftarrow pr3(Z,Y), pr2(X,Y).$
β) $pr2(X,Y) \leftarrow edb2(Z,Y), pr3(Z,X).$
γ) $pr3(X,Y) \leftarrow edb1(X), pr2(X,Y).$
δ) $pr3(X,Y) \leftarrow edb3(Y,X).$

Λόγω των κανόνων β και γ μεταξύ των κατηγορημάτων pr2 και pr3 υπάρχει σχέση αμοιβαία αναδρομικής εξάρτησης. Έτσι τα κατηγορήματα αυτά θα αποτελέσουν μία ομάδα κανόνων. Το κατηγόρημα pr1 εξαρτάται από τα δεδομένα των άλλων δύο και θα βρεθεί σε επόμενο γκρουπ μόνο του. Ο διαχωρισμός των κανόνων και οι προσωρινοί κανόνες που προκύπτουν για κάθε ομάδα κανόνων είναι:

Ομάδα 1 {β, γ, δ}
$pr3(X,Y) \leftarrow edb3(Y,X).$
$T_{i+1}pr2(X,Y) \leftarrow edb2(Z,Y), \Delta_i pr3(Z,X).$
$T_{i+1}pr3(X,Y) \leftarrow edb1(X), \Delta_i pr2(X,Y).$

Ομάδα 2 {α}
$pr1(X,Y) \leftarrow pr3(Z,Y), pr2(X,Y).$

Ο κανόνας δ δεν περιέχει κάποιο IDB κατηγόρημα στο σώμα του και δεν θα μεταφραστεί σε προσωρινό κανόνα, αλλά θα εκτελεστεί μια φορά πριν αρχίσει η επεξεργασία των υπόλοιπων προσωρινών κανόνων της ομάδας 1. Μόλις ολοκληρωθεί η αποτίμηση της πρώτης ομάδας τα κατηγορήματα pr2 και pr3 θα έχουν λάβει το σύνολο των πληροφοριών που μπορούν να εξαχθούν για αυτά έτσι κατά την αποτίμηση επόμενων ομάδων κανόνων θα θεωρούνται EDB. Για αυτό τον λόγο η δεύτερη ομάδα δεν περιέχει κάποιον προσωρινό κανόνα.

Αν δεν γίνει διαχωρισμός των κανόνων αυτών θα έχουμε τους παρακάτω κανόνες:

α) $T_{i+1}pr1(X,Y) \leftarrow \Delta_{i\_}pr3(Z,Y), S_{i-1}pr2(X,Y).$
$T_{i+1}pr1(X,Y) \leftarrow S_{i\_}pr3(Z,Y), \Delta_i pr2(X,Y).$
β) $T_{i+1}pr2(X,Y) \leftarrow edb2(Z,Y), \Delta_i pr3(Z,X).$
γ) $T_{i+1}pr3(X,Y) \leftarrow edb1(X), \Delta_i pr2(X,Y).$
δ) $pr3(X,Y) \leftarrow edb3(Y,X).$

Η διαφορά εκ πρώτης όψεως μπορεί να φαίνεται μικρή. Το πλήρες μέγεθος της εξοικονόμησης που επιτυγχάνεται με την ομαδοποίηση γίνεται αντιληπτό αν αναλογιστούμε ότι οι προσωρινοί κανόνες επαναλαμβάνονται μέχρι να σταματήσει η παραγωγή νέων

δεδομένων. Αν απαιτούνται n επαναλήψεις για την εξαγωγή της πληροφορίας των κατηγορημάτων της ομάδας 1 η διαφορά στον αριθμό των εκτελεσθέντων στη βάση κανόνων είναι 2n – 1. Η διαφορά διευρύνεται ακόμα περισσότερο αν συνυπολογίσουμε και τις εντολές ένωσης και διαφοράς που απαιτούνται για τον σχηματισμό των ενδιάμεσων αποτελεσμάτων κάθε κατηγορήματος για το οποίο υπάρχει προσωρινός κανόνας.

### *3.4.2 Ιδιαιτερότητες της υλοποίησης*

Πολλές φορές ζητείται να δοθεί απάντηση σε κάποιο συγκεκριμένο ερώτημα. Είναι πολύ πιθανό για την απάντηση να μην απαιτείται ο υπολογισμός ολόκληρης της πληροφορίας που υπονοείται από το πρόγραμμα datalog και τα δεδομένα που το συνοδεύουν. Με τη βοήθεια του συμπτυγμένου γράφου, οι κορυφές του οποίου αναπαριστούν ολόκληρες Ι.Σ.Σ., μπορούμε εύκολα να διακρίνουμε τις περιπτώσεις όπου απαιτείται ολόκληρη η υπονοούμενη πληροφορία. Αυτό θα συμβεί μόνο στην περίπτωση που στο ερώτημα περιλαμβάνονται κατηγορήματα από όλες τις Ι.Σ.Σ. που βρίσκονται σε κορυφές του συμπτυγμένου γράφου που δεν έχουν εισερχόμενες ακμές, δηλαδή από αυτές που πρέπει να υπολογιστούν τελευταίες. Αν κατά τη δημιουργία του γράφου συμπεριλάβουμε και το ερώτημα τότε ο γράφος θα πρέπει να σχηματίζει ένα ισχυρά συνεκτικό δένδρο με ρίζα την κορυφή που περιλαμβάνει το κατηγόρημα στην κεφαλή του ερωτήματος. Στις περιπτώσεις που δεν συντρέχουν οι παραπάνω συνθήκες μπορούμε να συντομεύσουμε ακόμα περισσότερο την διαδικασία αποτίμησης ερωτημάτων διατηρώντας μόνο τους απαραίτητους για την απάντηση κανόνες. Αυτή είναι μια λειτουργία που μπορεί να πραγματοποιηθεί παράλληλα με την ομαδοποίηση των κανόνων. Όταν επιθυμούμε την απόρριψη κανόνων που δε συνεισφέρουν στο ερώτημα η διαδικασία τροποποιείται ως εξής: Αρχικά προστίθεται το ερώτημα στο σύνολο των κανόνων της εισόδου. Επειδή το κατηγόρημα στην κεφαλή του ερωτήματος δεν βρίσκεται στο σώμα κάποιου κανόνα είναι βέβαιο ότι θα αποτελεί μία από τις κορυφές χωρίς εισερχόμενες ακμές. Οι επόμενες κορυφές θα περιλαμβάνουν τις ομάδες κανόνων οι οποίες συνεισφέρουν στην απάντηση. Αυτός είναι και ο λόγος που επιλέχθηκε για τις ακμές του γράφου η αντίθετη φορά από αυτήν που διαισθητικά θα ταίριαζε για την αναπαράσταση της ροής της πληροφορίας. Η συνάρτηση Tdfs θα κληθεί απευθείας με όρισμα την κορυφή του ερωτήματος και θα αφεθεί να τρέξει, μέσω της αναδρομικής κλίσης στο σώμα της, σε όσες κορυφές είναι προσπελάσιμες από αυτήν. Αυτές είναι και οι μόνες που χρειαζόμαστε για την απάντηση. Ο βρόχος for που διασφαλίζει ότι θα εξεταστούν όλες οι κορυφές του γράφου θα αφαιρεθεί. Με

αυτό τον τρόπο και χωρίς κάποιο επιπλέον κόστος απορρίπτονται οι κανόνες που δεν ενδιαφέρουν και επιτυγχάνεται η περεταίρω βελτίωση της απόδοσης.

Είναι πλέον σαφής ο λόγος που η φορά των ακμών του γράφου επιλέχθηκε να είναι αντίθετη από αυτή του μονοπατιού δεδομένων που ταυτίζεται με τη φορά της ροής πληροφορίας μεταξύ των κατηγορημάτων. Αν είχε επιλεγεί η αυτή η φορά δε θα ήταν δυνατή η απόρριψη των κανόνων που δε συμβάλλουν στην απάντηση ενός ερωτήματος με τόσο κομψό τρόπο. Θα έπρεπε να εισαχθούν επιπλέον έλεγχοι για μια διαδικασία που είναι δυνατό να επιτευχθεί μέσω της εκμετάλλευσης του παραπροϊόντος της διαδικασίας ομαδοποίησης των κανόνων.

## *3.5 Μαγική μεταγραφή*

Η τεχνική της από κάτω προς τα πάνω αποτίμησης προγραμμάτων datalog είναι ιδανική για την εξόρυξη όλης της υπονοούμενης από το πρόγραμμα πληροφορίας. Υπάρχουν όμως περιπτώσεις όπου απαιτείται η απάντηση σε κάποιο συγκεκριμένο ερώτημα. Κάτι τέτοιο επιτυγχάνεται αποδοτικά με την από πάνω προς τα κάτω προσέγγιση του αποτελέσματος. Ξεκινώντας από το προς απάντηση ερώτημα είναι πολύ εύκολο να διακρίνει κανείς τις σχετικές πληροφορίες και κανόνες και να κατευθύνει άμεσα τον υπολογισμό προς το επιθυμητό αποτέλεσμα, χωρίς να απαιτηθεί η εξόρυξη όλων τον γεγονότων που υπονοούνται. Όταν όμως κινούμαστε αντίθετα δεν είναι προφανές ποια γεγονότα πρέπει να υπολογιστούν. Παρόλα αυτά υπάρχει λύση. Η έρευνα έχει αναδείξει την τεχνική της «μαγικής μεταγραφής» (magic rewriting) που επιτρέπει τον από κάτω προς τα πάνω υπολογισμό της απάντησης σε ερώτημα χωρίς να δημιουργούνται περισσότερα γεγονότα από όσα θα είχαμε με την από πάνω προς τα κάτω προσέγγιση. Όπως φανερώνει και το όνομα, οι κανόνες ξαναγράφονται με βάση το ερώτημα και ως δια μαγείας η απάντηση χτίζεται χωρίς περιττούς υπολογισμούς. Δύο είναι τα κύρια χαρακτηριστικά της από πάνω προς τα κάτω προσέγγισης που οδηγούν στον περιορισμό της παραγόμενης πληροφορίας. Το πρώτο είναι η προώθηση των εξαρτήσεων που προκύπτουν από τις σταθερές που περιέχονται στο ερώτημα στους κανόνες του προγράμματος. Το δεύτερο είναι η απόρριψη κανόνων που αφορούν κατηγορήματα που δεν συνδέονται μέσω κάποιου κανόνα με αυτά του ερωτήματος. Τα παραπάνω θα γίνουν ευκολότερα αντιληπτά μέσω ενός παραδείγματος.

**Παράδειγμα 5**

Ερώτημα:

*Q (Y) ← grandmotherOf (a, Y).*

Δεδομένα για το κατηγόρημα *childOf* :

| | |
|---|---|
| a | c |
| a | d |
| b | d |
| b | e |
| c | f |
| d | g |
| d | h |
| e | i |

Κανόνες:

1) *parentOf (X, Y) ← childOf (Y, X)*

2) *grandmotherOf (X, Y) ← parentOf (X, Z), female (X), childOf (Y, Z)*

Το ερώτημα του παραδείγματος 5, *Q(Y) ← grandmotherOf(a, Y)*, ικανοποιείται από τα αντικείμενα που συνδέονται με το *a* μέσω της σχέσης *grandmotherOf*. Αναζητούμε δηλαδή τα εγγόνια του *a*.

Αν ακολουθήσουμε την από πάνω προς τα κάτω προσέγγιση θα δούμε ότι το πρώτο βήμα είναι η προώθηση της σταθεράς «*a*» από το ερώτημα ένα επίπεδο χαμηλότερα μέσω του κανόνα 2: *grandmotherOf(a, Y) ← parentOf(a, Z), female(a), childOf(Y, Z).* Στη συνέχεια περνάμε με αντίστοιχο τρόπο στον κανόνα 1: *parentOf(a, Y) ← childOf(Y, a)*. Βλέπουμε ότι από το αρχικό ερώτημα προκύπτουν νέα υποερωτήματα, τα *parentOf(a, Z)* και *childOf(Y, a)*. Μέσω αυτής της στοχευμένης διαδικασίας αναζήτησης θα παραχθούν αποκλειστικά γεγονότα που αφορούν τον γονιό *a*, τα οποία είναι και τα μόνα απαραίτητα για να απαντηθεί το συγκεκριμένο ερώτημα.

Αν επιλέξουμε να κινηθούμε αντίθετα και ξεκινήσουμε από τους κανόνες προς το ερώτημα, θα απαιτηθεί να τροποποιήσουμε τους αρχικούς κανόνες για να πετύχουμε το ίδιο αποτέλεσμα. Θα προωθήσουμε τον περιορισμό που επιβάλλει η σταθερά του ερωτήματος στον κανόνα 2, που αφορά το *grandmotherOf*, μέσω δύο μηχανισμών. Ο πρώτος είναι ο στολισμός των κανόνων. Στην συνέχεια οι κανόνες αυτοί θα μεταγραφούν στο σύνολο των κανόνων που θα αποτιμηθούν από κάτω προς τα πάνω.

### *3.5.1 Στολισμός*

Για να αναπαραστήσουμε τις εξαρτήσεις που προκύπτουν από τις σταθερές θα προχωρήσουμε στη δημιουργία νέων κατηγορημάτων. Θα διαφοροποιούνται από τα αρχικά ανάλογα με τους όρους που αντιστοιχούν σε δεσμευμένη μεταβλητή. Πιο συγκεκριμένα θα συμπληρώσουμε το όνομα του κατηγορήματος με μια συμβολοσειρά από "f" και "b" ανάλογα με το αν η μεταβλητή στην αντίστοιχη θέση είναι ελεύθερη ή επιβάλλεται κάποιος περιορισμός στις τιμές που μπορεί να λάβει. Η διαδικασία ξεκινάει από τα κατηγορήματα που περιλαμβάνονται στο ερώτημα. Στο παράδειγμά μας έχουμε το κατηγόρημα *grandmotherOf* με την πρώτη θέση δεσμευμένη και την δεύτερη ελεύθερη άρα θα κατασκευάσουμε το κατηγόρημα *grandmotherOf_bf*. Στη συνέχεια θα αναζητήσουμε τους κανόνες που αφορούν το κατηγόρημα που μόλις στολίσαμε και θα δημιουργήσουμε ένα νέο στολισμένο κανόνα για κάθε έναν. Η διαδικασία θα επαναλαμβάνεται για κάθε διαφορετικό στολισμό του κατηγορήματος που θα προκύψει. Εδώ θα επεκτείνουμε τον στολισμό στα κατηγορήματα του σώματος του δεύτερου κανόνα. Με δεδομένη τη δέσμευση της μεταβλητής *X* στην κεφαλή του κανόνα, θα δημιουργηθεί πρώτα το κατηγόρημα *parentOf_bf*. Στη συνέχεια, πέρα από την μεταβλητή *X*, δεσμευμένη θα θεωρείται για το υπόλοιπο του κανόνα και η μεταβλητή *Z* του ατόμου *parentOf(X, Z)*, αφού η τιμή της πλέον θα περιορίζεται από τα δεδομένα που υπάρχουν ή μπορεί να εξαχθούν για το κατηγόρημα *parentOf*. Το νέο σύνολο δεσμευμένων μεταβλητών θα δώσει τα κατηγορήματα *female_b* και *childOf_fb*. Συνεχίζουμε τη διαδικασία στολίζοντας τους κανόνες που έχουν στην κεφαλή τα κατηγορήματα για τα οποία έχουμε δημιουργήσει νέες στολισμένες εκδοχές. Στο παράδειγμά μας μένει να στολιστεί ο κανόνας 1, ο οποίος θα γίνει: *parentOf_bf(X, Y) ← childOf_bf(Y, X)*. Τα *female* και *childOf* είναι EDB κατηγόρημα και δεν υπάρχει κανόνας με κεφαλή αυτό για να στολιστεί. Για αυτό το λόγο θα παραλειφθεί ο στολισμός τους. Φυσικά οι μεταβλητές EDB ατόμων θα προστίθενται κανονικά στο σύνολο των δεσμευμένων κατά τη διαδικασία του στολισμού αφού τα άτομα αυτά, παρά τον τύπο τους, δεν παύουν να περιορίζουν τις τιμές των μεταβλητών τους σύμφωνα με τα υπάρχοντα δεδομένα για αυτά στη βάση.

Η στολισμένη εκδοχή του παραδείγματος 5:

Κανόνες:

1) *parentOf_bf (X, Y) ← childOf (Y, X).*

2) *grandmotherOf_bf (X, Y) ← parentOf_bf (X, Z), female (X), childOf (Y, Z).*

Ερώτημα:

*Q (Y) ← grandmotherOf_bf (a, Y).*

Είναι σαφές ότι ο στολισμός των κατηγορημάτων στο σώμα εξαρτάται από την σειρά με την οποία αυτά θα επιλεγούν. Δεν υπάρχει κάποιος περιορισμός στην επιλογή της σειράς, όλοι οι δυνατοί συνδυασμοί θα οδηγήσουν στην ίδια απάντηση. Τα μεγέθη που επηρεάζει η σειρά επιλογής των ατόμων του κανόνα κατά τον στολισμό είναι τα εξής:

- Ο όγκος της πληροφορίας που θα παραχθεί για τα νέα στολισμένα κατηγορήματα. Όσο περισσότερες ελεύθερες μεταβλητές περιλαμβάνονται σε ένα στολισμένο κατηγόρημα τόσο μεγαλύτερο τείνει να είναι το τμήμα της συνολικής του πληροφορίας που θα περιέχει. Ως συνολική πληροφορία θα ορίσουμε την πληροφορία που θα παραγόταν για αυτό το κατηγόρημα αν δεν εφαρμοζόταν η τεχνική ης μαγικής μεταγραφής. Αν οδηγηθούμε στην δημιουργία κατηγορήματος με όλες τις μεταβλητές του ελεύθερες τότε, λόγω της απουσίας περιορισμών, θα παραχθεί για αυτό η συνολική πληροφορία. Όσο περισσότερες μεταβλητές δεσμεύονται τόσο μεγαλύτερη είναι και η πιθανότητα να παραχθεί μικρότερο μέρος της ολικής πληροφορίας. Βέβαια σε κάθε περίπτωση το άνω όριο της παραγωγής της ολικής πληροφορίας παραμένει. Αν στο παράδειγμά μας δεν περιλαμβάνονταν στα αρχικά δεδομένα τα γεγονότα *childOf(b, d)* και *childOf(b, e)* το στολισμένο κατηγόρημα *grandmotherOf_bf*, παρά τον περιορισμό στην πρώτη του μεταβλητή, θα περιείχε την συνολική πληροφορία για το κατηγόρημα *grandmotherOf*. Η στρατηγική με τη μεγαλύτερη πιθανότητα να οδηγήσει σε μείωση της παραγόμενης πληροφορίας είναι να επιλέγονται τελευταία τα IDB κατηγορήματα κατά τον στολισμό του σώματος. Η επιλογή αυτή έχει ως αποτέλεσμα να αυξάνεται η πιθανότητα να εμφανίζουν δεσμευμένες μεταβλητές.
- Το πλήθος των στολισμένων κανόνων που θα προκύψουν. Για κάθε διαφορετικό στολισμό ενός IDB κατηγορήματος πρέπει να στολιστούν εκ νέου όλοι οι κανόνες που το αφορούν. Θα μπορούσαμε να αναζητήσουμε την σειρά που δημιουργεί τις λιγότερες δυνατές διαφοροποιήσεις στους στολισμούς αλλά οι δυνατοί συνδυασμοί αυξάνονται με τον αριθμό των κανόνων της εισόδου και των ατόμων που αυτοί περιέχουν καθιστώντας την αναζήτησή της αρκετά “ακριβή” σε υπολογιστικούς πόρους. Η βέλτιστη λύση, όσον αφορά στον αριθμό των κανόνων, είναι η μηδενική. Δηλαδή να μην γίνει στολισμός και απλώς να αφαιρεθούν όσοι κανόνες δεν συμβάλλουν στην παραγωγή πληροφορίας για τα κατηγορήματα που περιέχονται στο ερώτημα. Έτσι καταλήγουμε με ένα μόνο σετ κανόνων για κάθε κατηγόρημα.

Φυσικά αυτό έρχεται σε αντίθεση με την επιδίωξη για περιορισμό της παραγόμενης πληροφορίας.

Η επιλογή της σειράς γίνεται με γνώμονα την άμεση προώθηση των δεσμευμένων μεταβλητών της κεφαλής στα κατηγορήματα του σώματος και τη διατήρηση, στη συνέχεια, μιας συνεχούς αλυσίδας δεσμευμένων μεταβλητών. Η επιδίωξη της συνεχούς αλυσίδας δεσμευμένων μεταβλητών επηρεάζει τη μορφή της μεταγραφής όπως θα δούμε στη συνέχεια. Για αυτό σε κάθε βήμα επιλέγεται ένα άτομο που περιλαμβάνει κάποια από τις ήδη δεσμευμένες μεταβλητές. Αν δεν βρεθεί τέτοιο τότε επιλέγεται τυχαία το επόμενο άτομο. Αν βρεθούν περισσότερα του ενός άτομα με δεσμευμένες μεταβλητές, προτεραιότητα θα δοθεί σε άτομα με EDB κατηγορήματα. Με αυτό τον τρόπο δημιουργείται μια τάση να παράγονται IDB κατηγορήματα με μεγάλο ποσοστό δεσμευμένων μεταβλητών, πράγμα που έχει ευεργετική επίδραση στον όγκο της παραγόμενης πληροφορίας. Παράλληλα τείνουν να περιοριστούν και τα διαφορετικά σετ στολισμένων κανόνων που προέρχονται από το ίδιο αρχικό κατηγόρημα αφού γίνονται σπανιότερες οι ελεύθερες μεταβλητές.

Σε αυτό το σημείο έχουμε ένα νέο σύνολο κανόνων στο οποίο χαρτογραφείται η διάδοση των εξαρτήσεων που προκύπτουν από τις σταθερές του ερωτήματος. Ο τρόπος κατασκευής του οδηγεί αυτόματα στην απόρριψη κανόνων που δεν συμβάλλουν στην απάντηση αφού δημιουργούνται στολισμένοι κανόνες μόνο για τα κατηγορήματα τα οποία περιλαμβάνονται σε κάποιο από τα μονοπάτια πληροφορίας που έχουν ως άκρο ένα από τα κατηγορήματα που περιλαμβάνονται στο ερώτημα.

### *3.5.2 Μεταγραφή*

Το σύνολο των στολισμένων κανόνων δεν μπορεί να χρησιμοποιηθεί ως έχει. Μια προσπάθεια από κάτω προς τα πάνω αποτίμησης των νέων κανόνων σε αυτό το σημείο όχι μόνο δεν θα επέφερε καμία βελτίωση, αλλά θα οδηγούσε σε εσφαλμένη απάντηση. Δεν υπάρχει κάποιος τρόπος να μεταφερθεί πληροφορία από τα αρχικά κατηγορήματα προς τα στολισμένα. Η μόνη δυνατή απάντηση στο στολισμένο ερώτημα είναι η κενή. Το επόμενο βήμα στην διαδικασία είναι η μεταγραφή των στολισμένων κανόνων με στόχο την αυστηρά ελεγχόμενη τροφοδοσία κάθε στολισμένου κατηγορήματος μόνο με τα απολύτως απαραίτητα, για την δημιουργία της απάντησης, δεδομένα. Αυτό θα πραγματοποιηθεί μέσω ενός νέου κατηγορήματος εισόδου για κάθε IDB κατηγόρημα που συναντάμε στους στολισμένους κανόνες. Το κατηγόρημα εισόδου θα περιλαμβάνει τις δεσμευμένες μεταβλητές

του κατηγορήματος στο οποίο αντιστοιχεί. Ο έλεγχος της πληροφορίας που θα καταλήγει στα στολισμένα IDB κατηγορήματα επιβάλλεται με την προσθήκη των κατηγορημάτων εισόδου στους υπάρχοντες κανόνες. Έτσι οι κανόνες εφαρμόζονται μόνο για τις τιμές των δεσμευμένων μεταβλητών οι οποίες περιλαμβάνονται στα κατηγορήματα εισόδου. Για το παράδειγμά μας θα χρειαστούμε δύο κατηγορήματα εισόδου, το *input_grandmotherOf_bf* και το *input_parentOf_bf*. Οι στολισμένοι κανόνες θα τροποποιηθούν ως εξής:

*parentOf_bf(X, Y)* ← *input_parentOf_bf(X), childOf(Y, X).*
*grandmotherOf_bf(X, Y)* ← *input_grandmotherOf_bf(X), female(X), parentOf_bf(X, Z), childOf(Y, Z).*

Ποιες είναι όμως ακριβώς οι τιμές που πρέπει να εισάγουμε στα κατηγορήματα εισόδου και που μπορούν να αναζητηθούν; Την απάντηση θα την βρούμε εξετάζοντας και τον τρόπο που πλησιάζουμε στην απάντηση με την από πάνω προς τα κάτω προσέγγιση. Η διαδικασία ξεκινά από ένα αρχικό ερώτημα και μέσω των κανόνων του προγράμματος datalog, γεννά τα κατάλληλα υποερωτήματα μέσω των οποίων θα αναζητηθούν όλα τα επιμέρους τμήματα της απάντησης. Άρα το επόμενο βήμα είναι να εισάγουμε στα κατηγορήματα εισόδου τις τιμές που προκύπτουν για τις δεσμευμένες μεταβλητές από το ερώτημα και τα υποερωτήματα.

Το αρχικό ερώτημα του παραδείγματος είναι το *grandmotherOf(a, Y)*. Για να υπολογίσουμε την απάντηση σε αυτό με τους νέους κανόνες θα πρέπει να εκτελέσουμε τους κανόνες που έχουν στην κεφαλή το κατηγόρημα *grandmotherOf_bf* με την τιμή *a* στη δεσμευμένη μεταβλητή. Άρα πρέπει να προσθέσουμε το γεγονός *input_grandmotherOf_bf(a)* ← στο προς εκτέλεση πρόγραμμα. Τα γεγονότα αυτού του τύπου αποκαλούνται «σπόροι» διότι από αυτές τις αρχικές τιμές ξεκινά η αποτίμηση της μαγικής μεταγραφής.

Στη συνέχεια θα κωδικοποιήσουμε τα υποερωτήματα στο νέο σύνολο κανόνων. Αυτά προέρχονται από την εφαρμογή των κανόνων του προγράμματος με δεσμευμένες μεταβλητές. Οι εξαρτήσεις εδώ είναι έμμεσες και καθορίζονται από τα δεδομένα στα οποία εφαρμόζεται κάθε φορά ο κανόνας αλλά και την σειρά με την οποία εξετάζονται τα άτομα στο σώμα του. Όπως είδαμε κατά το στολισμό του σώματος, οι δεσμευμένες μεταβλητές κάθε ατόμου είναι είτε αυτές που εμφανίζονται σε προηγούμενα άτομα είτε είναι δεσμευμένες στην κεφαλή του κανόνα. Κατά συνέπεια, όταν φτάνει η στιγμή να εξεταστεί ένα άτομο στο σύνολο των δεσμευμένων μεταβλητών του έχουν ήδη αποδοθεί οι τιμές οι οποίες πρέπει να ελεγχθούν. Οι τιμές αυτές θα πρέπει να τροφοδοτηθούν στο κατηγόρημα εισόδου για να ενεργοποιηθούν οι κανόνες που αντιστοιχούν στο συγκεκριμένο κατηγόρημα και να προχωρήσει η παραγωγή πληροφορίας για αυτό. Ο πρώτος κανόνας του παραδείγματος δεν περιέχει κάποιο IDB κατηγόρημα στο σώμα άρα δεν επηρεάζεται. Ο δεύτερος κανόνας θα τροποποιηθεί όπως φαίνεται παρακάτω.

Η τελευταία μορφή του δεύτερου κανόνα είναι:

*grandmotherOf_bf(X, Y)* ← *input_grandmotherOf_bf(X), female(X), parentOf_bf(X, Z), childOf(Y, Z)*

Θα διασπαστεί σε δύο κανόνες για να μπορέσουμε να κατευθύνουμε τις δεσμευμένες μεταβλητές και στο κατηγόρημα εισόδου:

*sup(X)* ← *input_grandmotherOf_bf(X), female(X)*
*grandmotherOf_bf(X, Y)* ← *sup(X), parentOf_bf(X, Z), childOf(Y, Z)*

Τέλος, θα προστεθεί ένας ακόμα κανόνας για την ενημέρωση του κατηγορήματος εισόδου:

*input_parentOf_bf(X)* ← *sup(X)*

Το βοηθητικό κατηγόρημα *sup* δε θα διατηρεί μόνο τις τιμές των δεσμευμένων μεταβλητών που περιλαμβάνει το IDB κατηγόρημα που προκάλεσε τη δημιουργία του και απαιτείται να εισαχθούν στο αντίστοιχο κατηγόρημα εισόδου. Είναι αναγκαίο να διατηρηθούν και όσες δεσμευμένες μεταβλητές εμφανίζονται στα άτομα που ακολουθούν στο υπόλοιπο σώμα, για να μεταφερθούν στο επόμενο κομμάτι του κανόνα και να συνεχίσει απρόσκοπτα η αποτίμησή του. Επιπλέον, θα πρέπει να διατηρηθούν και όσες από τις μεταβλητές που βρίσκονται στην κεφαλή του κανόνα έχουν εμφανιστεί σε προηγούμενα άτομα, ακόμα και αν δεν χρησιμοποιούνται από κάποιο επόμενο άτομο στο σώμα. Αυτές οι μεταβλητές αποτελούν μέρος του τελικού αποτελέσματος της αποτίμησης του κανόνα και θα πρέπει να μεταβιβαστούν στο κατηγόρημα της κεφαλής από το τελευταίο τμήμα του.

Υπάρχει περίπτωση οι μεταβλητές που πρέπει να διατηρηθούν από ένα βοηθητικό κατηγόρημα να είναι περισσότερες από δύο. Λόγω του γεγονότος ότι ο μέγιστος αριθμός των μεταβλητών που μπορεί να υπάρξουν στα άτομα του προγράμματος datalog που δημιουργείται από το Rapid είναι δύο, τα αντικείμενα που επικοινωνούν με τις βάσεις, ανεξαρτήτως τύπου, είναι κατασκευασμένα με αυτό τον περιορισμό. Επίσης ο χειρισμός κατηγορημάτων με περισσότερους από δύο όρους σε RDF βάση αυξάνει την πολυπλοκότητα κατά την εκτέλεση των κανόνων διότι απαιτούνται σύνολα τριάδων για κάθε πλειάδα που πρέπει να αποθηκευτεί σε αυτά. Επειδή η ανάγκη για την ύπαρξη κατηγορημάτων με περισσότερους από δύο όρους εμφανίζεται μόνο σε αυτό το σημείο και για μικρό αριθμό πολύπλοκων κανόνων, κρίθηκε σκόπιμο να αντιμετωπιστεί με την αποτροπή της δημιουργίας βοηθητικού κατηγορήματος. Κατά συνέπεια σε αυτές τις περιπτώσεις δε σπάει ο κανόνας πριν από το IDB κατηγόρημα που είναι υπό εξέταση. Αντί για το βοηθητικό κατηγόρημα, στο σώμα του κανόνα που τροφοδοτεί το κατηγόρημα εισόδου τοποθετούνται απευθείας τα άτομα

που θα αποτελούσαν το σώμα του κανόνα για το βοηθητικό κατηγόρημα. Αυτό το τέχνασμα έχει το μειονέκτημα ότι στέλνεται εντολή για την αποτίμηση ενός μέρους του κανόνα δύο φορές. Παράλληλα όμως έχει θετική επίδραση στον αριθμό των απαιτούμενων κανόνων λόγω της αποφυγής της διάσπασης. Έτσι μειώνεται κατά ένα ο αριθμός των κανόνων που περιέχει η μαγική μεταγραφή. Ο τρόπος χειρισμού αυτής της περίπτωσης φαίνεται στο παράδειγμα 6.

Μέτρα για τον περιορισμό των σημείων όπου επιβάλλεται η μεταφορά πολλών δεσμευμένων μεταβλητών μπορούν να ληφθούν κατά τον στολισμό των κανόνων. Η τροποποίηση της σειράς των ατόμων στο σώμα για τη διατήρηση μιας συνεχούς αλυσίδας δεσμευμένων μεταβλητών, έχει ως συνέπεια τη μείωση του αριθμού των μεταβλητών αυτών αφού τείνουν να αξιοποιούνται από τα αμέσως επόμενα άτομα. Ακόμα, στις περιπτώσεις όπου υπάρχουν περισσότερα του ενός άτομα που πληρούν τα κριτήρια τις επιλογής, όπως αυτά αναφέρθηκαν προηγουμένως, μπορεί να επιλέγεται άτομο που δεν περιλαμβάνει μεταβλητή που υπάρχει στην κεφαλή του κανόνα. Επειδή αυτές οι μεταβλητές επιβάλλεται να διατηρηθούν κατά τη μεταγραφή μέχρι το τελευταίο τμήμα του κανόνα για να μεταφερθούν οι παραγόμενες πληροφορίες στο κατηγόρημα της κεφαλής, από την στιγμή που θα δεσμευτούν θα παραμείνουν δεσμευμένες ανεξάρτητα από το αν απαιτούνται από επόμενα άτομα στο σώμα.

Σε πολύπλοκα σύνολα κανόνων εμφανίζεται η ανάγκη για την διάκριση μεταξύ των διαφορετικών βοηθητικών κατηγορημάτων. Για αυτό το λόγο στα ονόματά τους χρησιμοποιείται η παρακάτω σύμβαση. Το όνομα sup θα ακολουθείται από έναν αριθμό που θα υποδηλώνει πόσα βοηθητικά κατηγορήματα έχουν δημιουργηθεί για τον συγκεκριμένο κανόνα. Για τον διαχωρισμό μεταξύ των κανόνων θα προστεθεί το "_r" ακολουθούμενο από τον αύξοντα αριθμό του στολισμένου κανόνα για τον οποίο δημιουργήθηκε.

**Παράδειγμα 6: Διατήρηση περισσότερων από δύο τιμών δεσμευμένων μεταβλητών.**

*hPr(X, Y) ← edb1(Z), idb1(Z, X1), edb2(Y, X1), idb2(X, X1), edb3(X).*

Για το υποερώτημα *hPr_bf(X, Y)*, $X \in \Omega$ με το $\Omega$ να αποτελεί ένα σύνολο τιμών που μπορεί να λάβει η δεσμευμένη μεταβλητή, ο κανόνας θα μεταγραφεί ως εξής:

*input_idb1_bf(Z)* ← ***input_hPr_bf(X), edb1(Z, U)***.
*sup1_r1(V, X)* ← ***input_hPr_bf(X), edb1(Z, U)***, *idb1_bf(Z, V), edb2(Y, U).*
*input_idb2_fb(V)* ← *sup1_r1(V, X).*
*hPr_bf(X, Y)* ← *sup1_r1(V, X), idb2_fb(X, V), edb3(X).*

Το υπογραμμισμένο τμήμα στους κανόνες για τα *input_idb1_bf* και *sup1_r1(V, X)* είναι κοινό. Αντιστοιχεί στο *sup0_r1(V, X, Z)* το οποίο λόγω της υπέρβασης στον αριθμό των μεταβλητών δεν δημιουργείται.

Ένα ακόμα σημείο όπου απαιτείται τροποποίηση της διαδικασίας της μεταγραφής είναι όταν το κατηγόρημα της κεφαλής εμφανίζεται στο σώμα του κανόνα έχοντας παράλληλα τον αυτό στολισμό και ταυτόχρονα συνοδεύεται από τις ίδιες δεσμευμένες μεταβλητές. Η ιδιαιτερότητα έγκειται στο ότι και στα δύο αυτά άτομα αντιστοιχεί το ίδιο κατηγόρημα εισόδου. Οι δυνατές τιμές των δεσμευμένων μεταβλητών για το συγκεκριμένο κατηγόρημα δεν μπορούν να αυξηθούν όσο λαμβάνουμε υπόψη μας νέα άτομα στο σώμα του κανόνα. Αυτό που μπορεί να συμβεί είναι να έχουμε μείωση των δυνατών τιμών λόγω των περιορισμών που εισάγουν τα νέα άτομα. Για αυτό το λόγο είναι περιττή η δημιουργία σε αυτό το σημείο κανόνα που θα εισάγει τις συγκεκριμένες τιμές στο κατηγόρημα εισόδου, αφού είναι βέβαιο ότι τις περιέχει ήδη. Την μεταγραφή ενός τέτοιου κανόνα βλέπουμε στο παράδειγμα 7.

Η τελευταία εξαίρεση στη διαδικασία της μεταγραφής ενός κανόνα έχει να κάνει με την περίπτωση όπου το βοηθητικό κατηγόρημα που δημιουργείται κατά την διάσπαση του κανόνα τροφοδοτείται αποκλειστικά από ένα κατηγόρημα. Αυτό έχει ως αποτέλεσμα να δημιουργείται ένα κατηγόρημα το οποίο είναι αντίγραφο του αρχικού. Φυσικά κάτι τέτοιο επιβαρύνει τη βάση δεδομένων χωρίς να παρέχει κανένα όφελος. Για την αντιμετώπιση αυτής της κατάστασης θα αποτραπεί η δημιουργία του βοηθητικού κατηγορήματος και τον ρόλο του θα επωμιστεί το αρχικό κατηγόρημα. Στο παράδειγμα 8 παρουσιάζεται η διαφοροποίηση αυτή.

**Παράδειγμα 7: Αποτροπή διάσπασης κανόνα.**

*Pred(X, Y) ← Pred(Z, Y), PredB(Z, X).*

Για το υποερώτημα *Pred_fb(Y)*, $X \in \Omega$ με το $\Omega$ να αποτελεί ένα σύνολο τιμών που μπορεί να λάβει η δεσμευμένη μεταβλητή, αν υποθέσουμε ότι και το *PredB* είναι IDB κατηγόρημα ο κανόνας θα μεταγραφεί ως εξής:

***sup1_r1(Y)*** ***← input_Pred_fb(Y).***
***input_Pred_fb(Y)*** ***← sup1_r1(Y).***
*sup2_r1(Y, Z)* *← sup1_r1(Y), Pred_fb(Z, Y).*
*input_PredB_bf(Z)* *← sup2_r1(Y, Z).*
*Pred_fb(X, Y)* *← sup2_r1(Y, Z), PredB_bf(Z, X).*

Οι δύο πρώτοι κανόνες της μεταγραφής δεν συνεισφέρουν σε κάποια χρήσιμη λειτουργία αφού ανακυκλώνουν τα ίδια ακριβώς δεδομένα στο κατηγόρημα εισόδου *input_Pred_fb*. Αυτό το φαινόμενο εμφανίζεται εδώ επειδή επαναλαμβάνεται το κατηγόρημα της κεφαλής, με τον ίδιο στολισμό, στην αρχή του σώματος του κανόνα. Άρα το κατηγόρημα εισόδου που αντιστοιχεί στον κανόνα είναι το ίδιο με το κατηγόρημα εισόδου του πρώτου κατηγορήματός του. Αν προχωρήσουμε κανονικά στην μεταγραφή αυτού του κανόνα, το κατηγόρημα εισόδου θα ανατροφοδοτηθεί με τα ίδια δεδομένα και είναι πασιφανές ότι θα έχουμε ένα ζευγάρι περιττών κανόνων. Έτσι καταλήγουμε, μέσω του πλεονάζοντος βοηθητικού κατηγορήματος, να ανακυκλώνουμε την ίδια πληροφορία. Η ανάγκη πίσω από τη δημιουργία βοηθητικών κατηγορημάτων όπου εμφανίζεται IDB κατηγόρημα σε σώμα κανόνα είναι η τροφοδοσία του κατηγορήματος εισόδου του τελευταίου με τις τιμές των δεσμευμένων μεταβλητών όπως προκύπτουν από τους περιορισμούς που επιβάλλουν τα προηγούμενα άτομα. Σε αυτή την περίπτωση παρακάμπτεται η διάσπαση του κανόνα για να αποφευχθεί ο πλεονασμός. Έτσι καταλήγουμε με τη μεταγραφή:

*sup1_r1(Y, Z)* ← *input_Pred_fb(Y), Pred_fb(Z, Y).*
*input_PredB_bf(Z)* ← *sup1_r1(Y, Z).*
*Pred_fb(X, Y)* ← *sup1_r1(Y, Z), PredB_bf(Z, X).*

Αν το κατηγόρημα της κεφαλής βρεθεί (με τον ίδιο στολισμό) στο σώμα του κανόνα αλλά το άτομο στο οποίο ανήκει περιέχει διαφορετικές μεταβλητές τότε ακολουθείται κανονικά η διαδικασία της μεταγραφής αφού οι τιμές των συγκεκριμένων δεσμευμένων μεταβλητών δεν περιορίζονται από το κατηγόρημα εισόδου στην αρχή του σώματος.

**Παράδειγμα 8: Απόρριψη βοηθητικού κατηγορήματος.**

Έστω ο παρακάτω κανόνας, με το κατηγόρημα *Pred3* να είναι EDB:

*Pred(X, Y)* ← *Pred2(Z, Y), Pred3(Z, X).*

Αν εμφανιστεί το υποερώτημα *Pred_ff(X, Y)* θα οδηγήσει στη δημιουργία του στολισμού: *Pred_ff(X, Y)* ← *Pred3(Z, X), Pred2_bf(Z, Y)*. Αυτός με τη σειρά του θα μεταγραφεί στους παρακάτω:

*sup1_r1(Z, X)* ← *Pred3(Z, X).*
*input_Pred2_bf(Z)* ← *sup1_r1(Z, X).*
*Pred_ff(X, Y)* ← *sup1_r1(Z, X), Pred2(Z, Y).*

Είναι πασιφανές ότι για το βοηθητικό κατηγόρημα *sup1_r1* δεν είναι δυνατό να περιέχονται στη βάση διαφορετικές πληροφορίες από αυτές που υπάρχουν για το *Pred3* καθώς οι συγκεκριμένοι κανόνες είναι και οι μοναδικοί που το αναφέρουν. Το *sup1_r1* αποτελεί ένα αντίγραφό του *Pred3*. Αποτρέποντας την δημιουργία του και χρησιμοποιώντας απευθείας το *Pred3* στη θέση του μειώνουμε το μέγεθος της μεταγραφής κατά έναν κανόνα και τελικά έχουμε:

*input_Pred2_bf(Z)* ← *Pred3(Z, X).*
*Pred_ff(X, Y)* ← *Pred3(Z, X), Pred2_bf(Z, Y).*

### *3.5.2.1 Ακραίες περιπτώσεις στολισμού*

Στην περίπτωση που προκύψει στολισμός όπου όλες οι μεταβλητές είναι ελεύθερες η διαδικασία της μεταγραφής πρέπει να διαφοροποιηθεί. Αν προσπαθήσουμε να δημιουργήσουμε κατηγόρημα εισόδου θα διαπιστώσουμε ότι δεν μπορεί να οριστεί λόγω της έλλειψης δεσμευμένων μεταβλητών. Εδώ δεν υπάρχουν περιορισμοί, όλα τα γεγονότα που είναι δυνατό να παραχθούν από τους διαθέσιμους κανόνες είναι αποδεκτά. Για αυτό δεν θα προστίθεται κάποιο κατηγόρημα εισόδου σε κανόνες στους οποίους το άτομο της κεφαλής έχει όλες τις μεταβλητές ελεύθερες. Παράλληλα δεν θα διαχωρίζεται κανόνας σε σημείο όπου υπάρχει IDB κατηγόρημα χωρίς δεσμευμένη μεταβλητή. Κατά τη μαγική μεταγραφή τα κατηγορήματα αυτά, όταν απαντώνται στο σώμα κανόνων, θα αντιμετωπίζονται όπως τα EDB κατηγορήματα.

Τα παραπάνω οδηγούν σε σκέψεις για την βελτίωση της μεταγραφής. Ίσως είναι δυνατό να αποτραπεί η εκτέλεση των κανόνων που αντιστοιχούν στο ίδιο κατηγόρημα, με δεσμευμένες όμως μεταβλητές και να λαμβάνονται οι πληροφορίες που ικανοποιούν τις δεσμεύσεις φιλτράροντας μέσω του αντίστοιχου κατηγορήματος εισόδου τις πληροφορίες που υπάρχουν για το κατηγόρημα με όλες τις μεταβλητές ελεύθερες. Ένα παράδειγμα αυτού του είδους κανόνα είναι το παρακάτω: *Pr_fb(X, Y)* ← *input_Pr_fb(Y), Pr_ff(X, Y).*

Το ερώτημα που πρέπει να απαντηθεί με σιγουριά για να χρησιμοποιηθεί αυτή η προσέγγιση είναι αν το κατηγόρημα με όλες τις μεταβλητές ελεύθερες περιλαμβάνει και την πληροφορία του ίδιου κατηγορήματος που υπόκειται σε κάποιες δεσμεύσεις. Η απάντηση είναι καταφατική αφού οι μόνοι περιορισμοί που υπάρχουν σε αυτή την περίπτωση είναι οι περιορισμοί που κωδικοποιούνται στα σώματα των κανόνων για το κατηγόρημα αυτό. Η απουσία κατηγορήματος εισόδου έχει άρει τους επιπρόσθετους περιορισμούς στην ροή πληροφορίας που υπάρχουν στους υπόλοιπους κανόνες της μεταγραφής. Έτσι είναι βέβαιο

ότι θα έχουμε το σύνολο της διαθέσιμης και υπονοούμενης πληροφορίας στα κατηγορήματα με ελεύθερες μεταβλητές. Άρα, όταν υπάρχει κάποιο κατηγόρημα χωρίς δεσμευμένη μεταβλητή μπορούμε να αφαιρέσουμε τους κανόνες που αφορούν εμφανίσεις του ίδιου κατηγορήματος με δεσμευμένες μεταβλητές και να αντλήσουμε την συμβατή με τις δεσμεύσεις πληροφορία με τον τρόπο που μόλις είδαμε. Όμως οι κανόνες στην μεταγραφή δεν επιτελούν μόνο τη λειτουργία της πλήρωσης με δεδομένα των κατηγορημάτων στην κεφαλή τους. Έχουν την επιπλέον την αποστολή να εισάγουν τις κατάλληλες τιμές στα κατηγορήματα εισόδου των IDB κατηγορημάτων που περιέχονται στο σώμα τους. Σε αυτό το σημείο τίθεται το ερώτημα αν είναι ασφαλές να καταργηθούν όλα τα τμήματα της μεταγραφής που αντιστοιχούν στους κανόνες που επιθυμούμε να αφαιρέσουμε ή θα πρέπει να απομακρύνουμε μόνο τα τελικά τμήματά τους που ως μόνο στόχο έχουν να τροφοδοτούν με πληροφορία το κατηγόρημα της κεφαλής. Η εισαγωγή τιμών στα διάφορα κατηγορήματα εισόδου από κάθε επιμέρους κανόνα που προκύπτει από τη μεταγραφή του αρχικού έχει ως μόνο στόχο τον έλεγχο της καταλληλότητας τους για την δημιουργία νέας πληροφορίας για το κατηγόρημα στην κεφαλή του. Γίνεται λοιπόν σαφές ότι επιβάλλεται να αφαιρεθούν όλοι οι κανόνες που δημιουργούνται κατά την μεταγραφή του αρχικού στολισμένου κανόνα για να επιτευχθεί η μέγιστη δυνατή εξοικονόμηση πόρων. Στο παράδειγμα 9 βλέπουμε τη βελτίωση της μεταγραφής στην πράξη.

**Παράδειγμα 9: Βελτίωση της μεταγραφής για άτομα χωρίς δεσμευμένες μεταβλητές.**

*Q(X)* ← *Pred(a, X).*
*Pr(X, Y)* ← *edbPr(X), Pr(Z, Y).*
*Pr(X, Y)* ← *edbPr2(X, Z), Pr(Z, Y).*

Μετά από τον στολισμό:

*Q(X)* ← *Pr_bf(a, X).*

*κανόνας 0: Pr_bf(X, Y)* ← *edbPr(X), Pr_ff(Z, Y).*
*κανόνας 1: Pr_bf(X, Y)* ← *edbPr2(X, Z), Pr_bf(Z, Y).*
*κανόνας 2: Pr_ff(X, Y)* ← *edbPr(X), Pr_ff(Z, Y).*
*κανόνας 3: Pr_ff(X, Y)* ← *edbPr2(X, Z), Pr_bf(Z, Y).*

Απλή μεταγραφή:

| | | |
|---|---|---|
| 0 | *Pr_bf(X, Y)* | ← *input_Pr_bf(X), edbPr(X), Pr_ff(Z, Y).* |
| 1 | *sup1_r1(X, Z)* | ← *input_Pr_bf(X), edbPr2(X, Z).* |
| | *input_Pr_bf(Z)* | ← *sup1_r1(X, Z).* |
| | *Pr_bf(X, Y)* | ← *sup1_r1(X, Z), Pr_bf(Z, Y).* |

| | | |
|---|---|---|
| *2* | *Pr_ff(X, Y)* | ← *edbPr(X), Pr_ff(Z, Y).* |
| *3* | *sup1_r3(X, Z)* | ← *edbPr2(X, Z).* |
| | *input_Pr_bf(Z)* | ← *sup1_r3(X, Z).* |
| | *Pr_ff(X, Y)* | ← *sup1_r3(X, Z), Pr_bf(Z, Y).* |

Βελτιωμένη μεταγραφή:

| | | |
|---|---|---|
| *0 & 1* | *Pr_bf(X, Y)* | ← *input_Pr_bf(X), Pr_ff(X, Y).* |
| *2* | *Pr_ff(X, Y)* | ← *edbPr(X), Pr_ff(Z, Y).* |
| *3* | *sup1_r3(X, Z)* | ← *edbPr2(X, Z).* |
| | *input_Pr_bf(Z)* | ← *sup1_r3(X, Z).* |
| | *Pr_ff(X, Y)* | ← *sup1_r3(X, Z), Pr_bf(Z, Y).* |

σπόρος: *input_Pr_bf(a)* ←

κανόνας ελέγχου της βάσης: *Pr_bf(X, Y)* ← *input_Pr_bf(X), Pr(X, Y).*

Παρατηρούμε ότι οι μεταγραφές των στολισμένων κανόνων 0 και 1 έχουν αντικατασταθεί από το φιλτράρισμα των δεδομένων για το *Pr_ff*. Οδηγούμαστε σε σημαντικά μικρότερη μεταγραφή η οποία υπολογίζει τα ίδια δεδομένα με αρκετά μικρότερο κόστος. Φυσικά αν ο κανόνας 3 δεν περιελάμβανε το *Pr_bf* θα προβαίναμε απλώς στην διαγραφή των κανόνων για αυτό.

Υπάρχει και η περίπτωση όλες οι μεταβλητές να είναι δεσμευμένες. Εδώ έχουμε ολική δέσμευση του IDB κατηγορήματος από τα γεγονότα που περιλαμβάνονται στο κατηγόρημα εισόδου που του αντιστοιχεί . Επειδή όλες οι μεταβλητές είναι δεσμευμένες δεν υπάρχει περιθώριο να προστεθεί άλλη πληροφορία από έναν κανόνα που αντιστοιχεί σε ένα τέτοιο κατηγόρημα. Αυτό ίσως μας βάλει σε πειρασμό να παρακάμψουμε τα κατηγορήματα εισόδου σε αυτήν την περίπτωση. Όμως κάτι τέτοιο δεν μπορεί να συμβεί γιατί έτσι προστίθενται οποιεσδήποτε πλειάδες προκύπτουν κατά την αποτίμηση τρίτων κανόνων χωρίς να λαμβάνονται υπόψη οι περιορισμοί των κανόνων που αφορούν το συγκεκριμένο κατηγόρημα. Άρα πρέπει να εκτελούνται κανονικά οι κανόνες και σε αυτές τις περιπτώσεις, όχι για την ανακάλυψη καινούριων τιμών αλλά για την επιβεβαίωση ότι τα δεδομένα των κατηγορημάτων εισόδου μπορούν να προέλθουν από αυτούς. Τότε και μόνο τότε αυτά μπορούν να προωθηθούν στα τελικά πλήρως δεσμευμένα κατηγορήματα.

### *3.5.3 Κανόνες για τον έλεγχο πρόσθετων δεδομένων στη βάση*

Στο περιβάλλον όπου καλείται να λειτουργήσει το πρόγραμμα δεν μπορεί να θεωρηθεί ότι πριν την εκτέλεσή του δεν υπάρχουν δεδομένα για τα IDB κατηγορήματα των δοσμένων κανόνων. Τα δεδομένα αυτά μπορεί να προϋπάρχουν στη βάση ή να δίνονται σαν γεγονότα μαζί με τους κανόνες. Λόγω της τροποποίησης των ονομάτων με την προσθήκη του στολισμού η μεταγραφή των κανόνων ως έχει δεν τα λαμβάνει υπόψη. Η λύση της καθολικής ενσωμάτωσης όλης της πληροφορίας θα είχε ως συνέπεια την παραβίαση της κατ' απαίτησης εξέτασης συγκεκριμένων τιμών για τις δεσμευμένες μεταβλητές και την παραγωγή μόνο των γεγονότων που προκύπτουν από αυτές τις τιμές. Το να συμπεριληφθούν περισσότερα δεδομένα για κάποιο κατηγόρημα από αυτά που θα υπολογίζονταν μπορεί να μην φαίνεται σαν κακή ιδέα, ειδικά αν αναλογιστεί κανείς ότι παρέχονται χωρίς να υπάρχει η επιβάρυνση του υπολογισμού τους. Όμως κάτι τέτοιο θα μπορούσε να οδηγήσει στον υπολογισμό πολλών περισσότερων γεγονότων για κατηγορήματα που βρίσκονται παρακάτω από αυτά στο μονοπάτι ροής της πληροφορίας ακυρώνοντας την κεντρική ιδέα πίσω από την τεχνική της μαγικής μεταγραφής. Είναι απαραίτητο να στηθεί ένας μηχανισμός για την επιλεκτική ενσωμάτωσή των έτοιμων πληροφοριών.

Για το σκοπό αυτό θα δημιουργήσομε ένα νέο είδος κανόνων και θα αφήσουμε τον αλγόριθμο αποτίμησης να ελέγχει την ύπαρξη γεγονότων με τις κατάλληλες τιμές στις δεσμευμένες μεταβλητές. Οι τιμές αυτές βρίσκονται στο κατηγόρημα εισόδου, το οποίο θα συνδυαστεί στον κανόνα ελέγχου της βάσης με ένα άτομο που θα περιλαμβάνει το αρχικό κατηγόρημα χωρίς προσθήκες στο όνομα. Οι κανόνες ελέγχου της βάσης για το παράδειγμα 5 είναι:

*grandmotherOf_bf(X, Y)* ← *input_grandmotherOf_bf(X), grandmotherOf(X, Y).*

*parentOf_bf(X, Y)* ← *input_parentOf_bf(X), parentOf(X, Y).*

Με αυτό τον τρόπο διασφαλίζεται ο τακτικός έλεγχος των έτοιμων δεδομένων στη βάση. Τα παραπάνω ισχύουν για στολισμούς που περιλαμβάνουν και ελεύθερη και δεσμευμένη μεταβλητή. Οι ακραίες περιπτώσεις στολισμού δεν απαιτούν τέτοιου είδους κανόνες για διαφορετικούς λόγους η κάθε μία.

Για τον πλήρως δεσμευμένο στολισμό ο έλεγχος για δεδομένα στη βάση είναι περιττός αφού όταν προκύπτουν νέες τιμές στο αντίστοιχο κατηγόρημα εισόδου αρκεί ο έλεγχος μέσω των κανόνων της μεταγραφής για να καθοριστεί αν θα περιληφθούν ή όχι. Οι τιμές των μεταβλητών εξαρτώνται πλήρως από τις τιμές που προκύπτουν κατά την αποτίμηση κανόνων.

Το να ελεγχθεί και η βάση είναι άσκοπο αφού δεν έχει να προσφέρει κάποια επιπλέον πληροφορία.

Στην περίπτωση του ελεύθερου στολισμού τα πράγματα είναι επίσης απλά. Λόγω της απουσίας περιορισμών όλα τα γεγονότα πρέπει να ενσωματωθούν στο κατηγόρημα. Αυτό γίνεται πιο αποδοτικά με την ενσωμάτωση των υπαρχόντων στη βάση γεγονότων στα αντίστοιχα στολισμένα κατηγορήματα μία φορά προτού αρχίσει η αποτίμηση των κανόνων.

Σε αυτό το σημείο η μεταγραφή των κανόνων έχει ολοκληρωθεί. Η από κάτω προς τα πάνω αποτίμηση των νέων κανόνων θα οδηγήσει στην απάντηση του ερωτήματος μέσω του υπολογισμού μόνο όσων γεγονότων θα απαιτούνταν από την από πάνω προς τα κάτω προσέγγιση.

# 4

# *Η υλοποίηση*

Το πρόγραμμα σχεδιάστηκε με κύριο στόχο την ευελιξία στην χρήση και την επεκτασιμότητα. Κύριο χαρακτηριστικό είναι η ευκολία με την οποία μπορούν να υποστηριχτούν νέες τεχνολογίες αποθήκευσης δεδομένων. Με την υλοποίηση μιας διεπαφής που περιλαμβάνει μερικές απλές εντολές προς τον διαχειριστή δεδομένων μπορεί να υποστηριχτεί ένας νέος τρόπος αποθήκευσης δεδομένων. Με αυτό τον τρόπο είναι εφικτό να καλυφθούν ανάγκες που ξεπερνούν κατά πολύ τις βάσεις ενός ερευνητικού εργαστηρίου. Η ανάπτυξη έγινε στην γλώσσα Java. Στην ίδια γλώσσα είναι γραμμένο και το Rapid, συνεπώς η συνεργασία με αυτό είναι άμεση χωρίς να προκύπτουν νέες απαιτήσεις από τους υπάρχοντες χρήστες του εργαλείου αυτού. Επιπλέον, η Java επιτρέπει την χρήση της εφαρμογής χωρίς αλλαγές στα περισσότερα γνωστά υπολογιστικά περιβάλλοντα. Μια άλλη κομβικής σημασίας απόφαση ήταν η αποφυγή ανάσυρσης των δεδομένων της εκάστοτε βάσης. Αντιθέτως, η αποτίμηση επιλέχθηκε να γίνεται εντός της βάσης, επιτυγχάνοντας δύο στόχους. Πρωτίστως, επιτυγχάνεται ο περιορισμός στο ελάχιστο των απαιτήσεων υλικού από το σύστημα του χρήστη. Η εργασία με τη βάση γνώσης μπορεί να πραγματοποιηθεί από τους περισσότερους φορητούς υπολογιστές ή ακόμα και τάμπλετ καταναλώνοντας ελάχιστη ποσότητα κύριας μνήμης και χωρίς να επηρεάζεται από τη δυνητικά χαμηλή ταχύτητα της δευτερεύουσας μνήμης των συστημάτων αυτών. Επιπροσθέτως, η απομόνωση των δεδομένων στην αρχική τους θέση μέσω της εκτέλεσης των διεργασιών που απαιτούν μεγάλο όγκο πληροφοριών απευθείας από το σύστημα διαχείρισής τους, καθιστά την εγγύτητα στα δεδομένα περιττή. Οι απαιτήσεις επικοινωνίας περιορίζονται στο μέγιστο δυνατό βαθμό με

αποτέλεσμα να είναι δυνατή η εργασία ακόμα και μέσω ασύρματων δικτύων προσωπικών επικοινωνιών. Οι εντολές προς τη βάση δεδομένων έχουν περιορισμένη έκταση και η μοναδική λειτουργία που ενδεχομένως απαιτήσει μεγάλο όγκο δεδομένων για να πραγματοποιηθεί, είναι η ανάκτηση της απάντησης στην περίπτωση που τεθεί κάποιο ερώτημα. Επειδή αποφασίστηκε να εκτελούνται οι ενδιάμεσοι υπολογισμοί χρησιμοποιώντας τη βάση, απαιτείται ένας τρόπος να διαχωριστούν τα δεδομένα τα οποία δεν προορίζονται για αποθήκευση σε αυτήν. Αυτά, σε κάθε περίπτωση είναι τα ενδιάμεσα αποτελέσματα ή τα βοηθητικά κατηγορήματα που δημιουργούνται από τη μαγική μεταγραφή αλλά μπορεί να είναι και τα τελικά αν επιλεγεί να επανέλθει η βάση στην αρχική της κατάσταση μετά την απάντηση κάποιου ερωτήματος. Για το διαχωρισμό των δεδομένων που εγράφησαν από το πρόγραμμα, χρησιμοποιείται σαν αναγνωριστικό μία συμβολοσειρά που προστίθεται στο όνομα του κατηγορήματος. Η λύση της διατήρησης λίστας με τα ονόματα των κατηγορημάτων που πρέπει να διαγραφούν απορρίφθηκε για δύο λόγους. Πρώτον, απαιτεί πολλαπλούς ελέγχους για επικαλύψεις ονομάτων. Αντιθέτως το χτίσιμο της συμβολοσειράς, όπως θα δούμε αναλυτικά παρακάτω, γίνεται με τον αποδοτικότερο δυνατό τρόπο καθώς ελέγχεται μία μόνο φορά κάθε όνομα κατηγορήματος. Ο δεύτερος λόγος είναι ότι εξαιτίας της θέσης της στα ονόματα των κατηγορημάτων, η συμβολοσειρά μπορεί να αναγνωριστεί εύκολα και να χρησιμοποιηθεί για τον καθαρισμό της βάσης μετά από απρόοπτη διακοπή της διαδικασίας αποτίμησης. Για να μπορέσει να επιτελέσει τον σκοπό της, η συμβολοσειρά δεν θα πρέπει να περιέχεται στο όνομα κάποιου κατηγορήματος που βρίσκεται είτε στην βάση είτε στην κεφαλή κάποιου κανόνα. Για αυτού του είδους τα κατηγορήματα ενδέχεται να παραχθεί πληροφορία, η οποία θα πρέπει να μπορεί να διακριθεί από την ήδη υπάρχουσα ή προοριζόμενη για μόνιμη διατήρηση στη βάση.

Με τον τρόπο που μόλις είδαμε μπορούν εύκολα να διαγραφούν όσες πληροφορίες χρειάζεται. Το τι θα διαγραφεί καθορίζεται από τον τρόπο με τον οποίο θα χρησιμοποιηθεί η βάση. Μπορεί να ενσωματωθεί η νέα πληροφορία στο σύνολό της, συμπεριφορά που ακολουθείται αν δε δοθούν διαφορετικές οδηγίες. Αν επιθυμούμε το γρήγορο στήσιμο της βάσης για εκτέλεση μιας σειράς ερωτημάτων στα ίδια δεδομένα, μπορούμε να αφήσουμε μόνο τα γεγονότα που περιλαμβάνει το πρόγραμμα datalog της εισόδου. Μία ακόμα επιλογή είναι να επιστραφεί η βάση στην αρχική της κατάσταση μετά την αποτίμηση κάποιου ερωτήματος. Τέλος, μπορεί να διαγραφεί κάθε πληροφορία ή να αφεθούν στη βάση ακόμα και τα βοηθητικά κατηγορήματα για την εποπτεία της διαδικασίας αποτίμησης.

Η εφαρμογή, πέρα από τη χρήση της απευθείας από άλλα προγράμματα, μπορεί να λειτουργήσει και αυτόνομα για την αποτίμηση προγραμμάτων datalog. Δίνεται έτσι η δυνατότητα υποστήριξης επαγωγικών χαρακτηριστικών από βάσεις που δεν υποστηρίζουν αυτή την τεχνολογία.

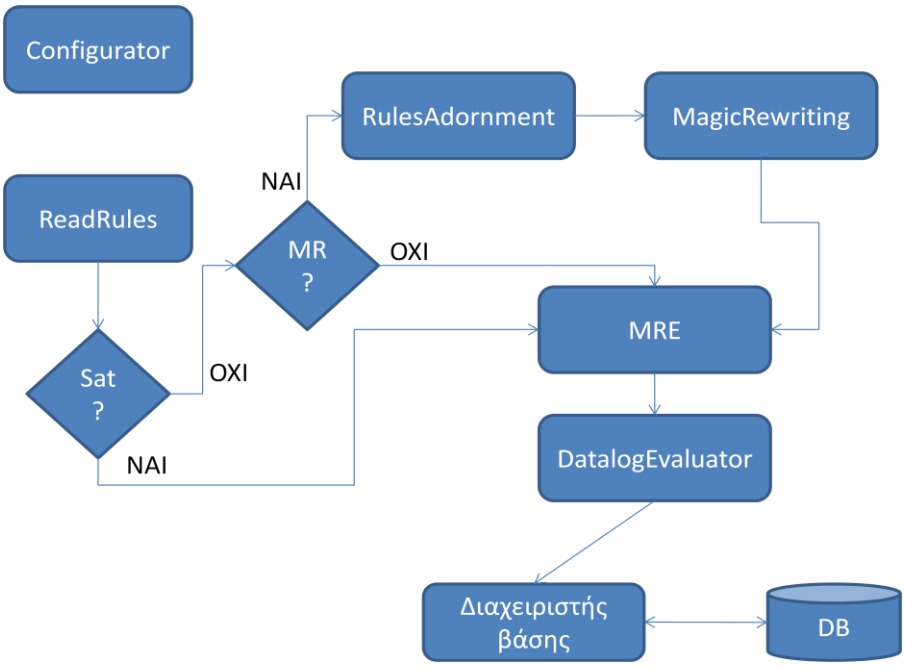
Το πρόγραμμα μπορεί να χρησιμοποιηθεί είτε ως API, απευθείας από άλλα προγράμματα όπως το Rapid, είτε ως αυτόνομη εφαρμογή. Σε κάθε περίπτωση, πρέπει να δοθούν ορισμένες πληροφορίες, όπως τα στοιχεία πρόσβασης στη βάση και ο τύπος της και η μεταγραφή που θα αποτιμηθεί. Υπάρχει πλήθος παραμέτρων που μπορούν να τροποποιηθούν, στις οποίες θα αναφερθούμε στη συνέχεια. Όλες οι παράμετροι συλλέγονται σε ένα αντικείμενο της κλάσης Configurator. Αυτό συμβουλεύονται όλα τα υποσυστήματα για να προσαρμόσουν τη συμπεριφορά τους σύμφωνα με τα ζητούμενα. Πρώτη κλάση που καλείται από την κύρια κλάση του προγράμματος είναι η ReadRules, η οποία αναλαμβάνει την εισαγωγή των κανόνων από την δοθείσα μεταγραφή. Στη συνέχεια υπάρχουν δύο επιλογές. Μπορεί να γίνει κορεσμός (saturation) της βάσης. Αυτό σημαίνει ότι θα εκτελεστούν όλοι οι κανόνες που περιλαμβάνει η μεταγραφή για να λάβουμε το σύνολο της υπονοούμενης πληροφορίας από την μεταγραφή με βάση τα αποθηκευμένα δεδομένα. Αυτό έχει ως αποτέλεσμα την άμεση αποστολή των κανόνων στην κλάση MutualRecursionEquivalence που αναλαμβάνει την ομαδοποίησή τους σύμφωνα με τα όσα περιγράφηκαν στο κεφάλαιο της datalog. Φυσικά, δεν πραγματοποιείται απόρριψη κανόνων σε αυτή την περίπτωση, ακόμα και αν έχει δοθεί κάποιο ερώτημα. Αν δεν επιθυμούμε τον κορεσμό της βάσης, έχουμε την επιλογή να χρησιμοποιήσουμε την τεχνική της μαγικής μεταγραφής (magic rewriting). Η χρήση της μαγικής μεταγραφής πρέπει να γίνεται με προσοχή διότι ενδέχεται να οδηγήσει και σε επιβάρυνση στο χρόνο εκτέλεσης. Η μαγική μεταγραφή των κανόνων της εισόδου έχει ως αποτέλεσμα την αύξηση του πλήθους τους. Συνέπεια αυτού είναι η αύξηση του αριθμού των κανόνων που πρέπει να εκτελεστούν στη βάση, πράγμα που φυσικά έχει ένα κόστος. Το κόστος αυτό αντισταθμίζεται από τα κέρδη που έχουμε λόγω του περιορισμού των νέων δεδομένων που δημιουργούνται αφού οι περιορισμοί του ερωτήματος λαμβάνονται υπόψη σε κάθε βήμα. Για το λόγο αυτό, η χρήση της μαγικής μεταγραφής ενδείκνυται όταν εργαζόμαστε με ιδιαίτερα μεγάλα σύνολα δεδομένων. Αν επιλέξουμε να την χρησιμοποιήσουμε αναλαμβάνουν δράση οι κλάσεις RulesAdornment και MagicRewriting. Η πρώτη θα αναλάβει τον στολισμό των κανόνων και στη συνέχεια η δεύτερη θα προχωρήσει στην μεταγραφή τους.

Στη συνέχεια, είτε πραγματοποιήθηκε η μαγική μεταγραφή είτε όχι προχωράμε στην τμηματοποίηση των κανόνων. Αυτή η τεχνική βελτιστοποίησης εφαρμόζεται πάντα διότι δεν είναι δυνατό να εισάγει καθυστερήσεις.

Μετά τον διαχωρισμό των κανόνων αναλαμβάνει δράση ο DatalogEvaluator, που είναι υπεύθυνος για την εκτέλεση της αποτίμησης. Δημιουργεί τους προσωρινούς κανόνες και τις εντολές ενημέρωσης και ερωτημάτων που απαιτούνται και τα μεταβιβάζει στον διαχειριστή της βάσης. Αυτός με τη σειρά του επικοινωνεί με τη βάση. Μεταφράζει τους κανόνες datalog και τις εντολές χειρισμού δεδομένων που λαμβάνει από τον DatalogEvaluator σε εντολές που

κατανοεί ο εξυπηρετητής της βάσης και επιστρέφει τα αποτελέσματα όπου υπάρχουν. Ανάλογα με τη χρησιμοποιούμενη βάση αλλάζει και η κλάση του διαχειριστή.

**Εικόνα 2: Διάγραμμα ροής της διαδικασίας αποτίμησης.**




Στη συνέχεια, ακολουθεί η περιγραφή διαφόρων τμημάτων του προγράμματος. Ως γνωστόν ο γραπτός λόγος δεν αποτελεί το ιδανικό μέσο για την παρουσίαση κώδικα. Ο αναγνώστης που ενδιαφέρεται για μια βαθύτερη γνωριμία με το σύστημα καλείται να συμβουλευτεί τη λεπτομερή τεκμηρίωση που το συνοδεύει.

## *4.1 Αποτιμητής datalog*

Η κλάση DatalogEvaluator αναλαμβάνει να φέρει σε πέρας την αποτίμηση του προγράμματος datalog. Υλοποιεί τον αλγόριθμο που παρουσιάστηκε παραπάνω για την από κάτω προς τα πάνω προσέγγιση του αποτελέσματος. Ξεκινά δηλαδή με τα αρχικά γνωστά γεγονότα και προχωράμε σταδιακά προς την εκμαίευση όλων των γεγονότων που μπορούν να προκύψουν από την εφαρμογή των κανόνων της μεταγραφής σε αυτά. Ανάλογα με τον τύπο της βάσης πάνω στην οποία θα εργαστεί δημιουργεί ένα αντικείμενο από την κατάλληλη κλάση για τον χειρισμό της και διαβιβάζει σε αυτό τις απαραίτητες εντολές.

Σαν είσοδο δέχεται τα εξής:

- Τους κανόνες datalog όπως ομαδοποιήθηκαν από την κλάση MutualRecursionEquivalence που πραγματοποιεί την τμηματοποίηση του προγράμματος όπως αναλύθηκε σε προηγούμενο κεφάλαιο. Η ομαδοποίηση φυσικά δεν είναι απαραίτητη, μπορούν να δοθούν όλοι οι κανόνες σαν μία ομάδα. Το αποτέλεσμα θα είναι το ίδιο αλλά θα φτάσουμε σε αυτό με λιγότερο αποδοτικό τρόπο.
- Το ερώτημα, αν έχει τεθεί.
- Τα γεγονότα που τυχόν περιείχε η είσοδος, τα οποία διαχωρίζονται από τους κανόνες από την ReadRules που αναλαμβάνει την εισαγωγή του προγράμματος datalog.
- Στην περίπτωση που έχει προηγηθεί μαγική μεταγραφή απαιτούνται επιπλέον δύο παράμετροι:
    - Μία δομή με τους κανόνες που είναι απαραίτητοι για τον έλεγχο της βάσης για ύπαρξη επιπλέον δεδομένων για τα IDB κατηγορήματα (dbLookupRules). Οι κανόνες αυτοί είναι επίσης διαχωρισμένοι σε ομάδες, αντίστοιχες με αυτές των κανόνων του προγράμματος datalog.
    - Τα ονόματα των αρχικών κατηγορημάτων για να είναι δυνατός ο διαχωρισμός τους από τα βοηθητικά κατηγορήματα της μαγικής μεταγραφής.
- Ένα αντικείμενο της κλάσης Configurator που περιέχει τις απαραίτητες παραμέτρους για την εκτέλεση.

Αρχικά δημιουργείται το κατάλληλο αντικείμενο χειρισμού της βάσης ανάλογα με το αν αυτή είναι τύπου RDF ή SQL. Κατόπιν ελέγχεται αν έχει προηγηθεί μαγική μεταγραφή στους κανόνες της εισόδου και αν αυτό ισχύει ακολουθούν δύο επιπλέον έλεγχοι για την ορθή χρήση του API. Αρχικά ελέγχεται αν έχουν δοθεί κανόνες που αξιοποιούν υπάρχουσες πληροφορίες στη βάση για IDB κατηγορήματα (dbLookupRules). Προειδοποιεί τον χρήστη για την απουσία των κανόνων και την πιθανότητα να μην ληφθούν υπόψη αυτές οι πληροφορίες. Ο δεύτερος έχει να κάνει με την απουσία του συνόλου με τα αρχικά ονόματα των κατηγορημάτων IDB. Αφού έχει χρησιμοποιηθεί η τεχνική της μαγικής μεταγραφής οι κανόνες της εισόδου έχουν τροποποιηθεί. Ως συνέπεια αυτής της διαδικασίας έχουν εμφανιστεί επιπλέον βοηθητικά IDB κατηγορήματα τα οποία πρέπει στο τέλος να αφαιρεθούν από την βάση αφού δεν περιλαμβάνουν κάποιο χρήσιμο αποτέλεσμα. Αυτό γίνεται ελέγχοντας αν το όνομα του κατηγορήματος περιέχεται στη δοσμένη λίστα με τα αρχικά IDB

κατηγορήματα. Αν υπάρχει η απαίτηση να ενσωματωθούν στη βάση οι πληροφορίες που θα παραχθούν κατά την διαδικασία υπολογισμού της απάντησης στο ερώτημα τότε θα δοθεί μήνυμα λάθους και το πρόγραμμα θα τερματίσει. Δεν μπορεί να προχωρήσει σε εισαγωγή νέας πληροφορίας στην βάση διότι δεν είναι δυνατό να διαχωριστούν με απόλυτη βεβαιότητα τα βοηθητικά κατηγορήματα που εισάγει η τεχνική της μαγικής μεταγραφής από τα αρχικά κατηγορήματα IDB των κανόνων, τα οποία και έχει ζητηθεί να διατηρηθούν.

Μετά τους ελέγχους εισάγονται στη βάση γεγονότα που τυχόν υπάρχουν στο αρχείο των κανόνων της εισόδου. Απλή εισαγωγή γίνεται σε όλες τις περιπτώσεις χειρισμού της βάσης εκτός και αν επιθυμούμε να επιστρέψει στην αρχική της κατάσταση μετά τον υπολογισμό της απάντησης στο ερώτημα. Για να είμαστε σε θέση να το κάνουμε αυτό θα πρέπει να διερευνήσουμε την ύπαρξη κάθε γεγονότος στη βάση, πράγμα που επιτυγχάνεται με τη χρήση της μεθόδου isAlreadyIn του διαχειριστή της βάσης. Αν αυτό ήδη υπάρχει τότε απλώς αφαιρείται από τη συλλογή με τα γεγονότα. Αν δεν υπάρχει τότε εισάγεται κανονικά, χωρίς κάποιο πρόθεμα, διατηρείται όμως στην συλλογή. Μόλις ολοκληρωθεί η διαδικασία αποτίμησης του ερωτήματος οι πληροφορίες που περιέχονται στα εναπομείναντα στη συλλογή γεγονότα θα αφαιρεθούν από τη βάση.

Στη συνέχεια αναζητείται από το αντικείμενο Configurator ο αριθμός των επεξεργαστών που είναι διαθέσιμοι για τις ανάγκες της κλάσης και αν αυτοί είναι περισσότεροι του ενός υπάρχει η δυνατότητα να ενεργοποιηθεί η παράλληλη δημιουργία προσωρινών κανόνων. Τέλος υπολογίζεται μέσω της κλάσης UniqueStringCreator η συμβολοσειρά που θα επισημαίνει τα προσωρινά δεδομένα στη βάση, το Unique Processing Prefix (UPP).

Σε αυτό το σημείο είναι όλα έτοιμα για την αποτίμηση των κανόνων στην βάση. Τα παρακάτω βρίσκονται σε ένα βρόχο που επαναλαμβάνεται για όλες τις ομάδες. Ο βρόχος ξεκινά με τον έλεγχο των κανόνων για το αν περιέχουν στο σώμα κάποιο κατηγόρημα IDB προηγούμενης ομάδας. Αν δεν προβλέπεται η ενσωμάτωση της παραγόμενης πληροφορίας στη βάση, οπότε και δεν εργαζόμαστε με τα τελικά ονόματα, οι πληροφορίες για αυτά έχουν αποθηκευτεί με το UPP. Για αυτό το λόγο το όνομα κάθε τέτοιου κατηγορήματος πρέπει να τροποποιηθεί. Παράλληλα επισημαίνονται τα κατηγορήματα στην κεφαλή των κανόνων της τρέχουσας ομάδας ως κατηγορήματα «IDB υπό επεξεργασία». Ο όρος «IDB υπό επεξεργασία» αποτελεί ουσιαστικά εξειδίκευση του όρου IDB εντός των ορίων μίας ομάδας κανόνων και όχι στο σύνολο των κανόνων της εισόδου. Περιλαμβάνει δηλαδή μόνο τα IDB κατηγορήματα της συγκεκριμένης ομάδας κανόνων. Ο αλγόριθμος αποτίμησης προγραμμάτων datalog όπως περιγράφηκε παραπάνω εφαρμόζεται ξεχωριστά σε κάθε ομάδα και έτσι σε ότι αφορά αυτόν ο όρος «IDB υπό επεξεργασία» ταυτίζεται με τον όρο IDB, όπως χρησιμοποιήθηκε κατά την περιγραφή του αλγορίθμου. Στο υπόλοιπο του κεφαλαίου, για συντομία, θα χρησιμοποιηθεί ο απλός όρος IDB. Αφού καταγραφούν τα κατηγορήματα IDB

αναζητούνται κανόνες της ομάδας που δεν περιέχουν κάποιο από αυτά στο σώμα. Οι κανόνες αυτοί, από τη στιγμή που δεν περιλαμβάνουν στο σώμα τους κάποιο κατηγόρημα για το οποίο θα παραχθεί πληροφορία, θα εκτελεστούν μια φορά προτού εισέλθουμε στον κύριο βρόχο του αλγορίθμου. Αν δεν υπάρχει κανόνας με κατηγόρημα IDB στο σώμα σε αυτό το σημείο υπάρχει μια συντόμευση. Παραλείπεται ο βρόχος υπολογισμού του σταθερού σημείου και οι πληροφορίες που παράγουν οι κανόνες αποθηκεύονται απευθείας με την τελική τους μορφή. Δηλαδή με το όνομα του κατηγορήματος αν εργαζόμαστε με τα τελικά ονόματα, διαφορετικά προστίθεται το UPP. Αν δεν εργαζόμαστε με τα τελικά ονόματα θα πρέπει να εκτελεστούν με αντίστοιχο τρόπο και οι dbLookupRules που τυχόν υπάρχουν για να ενσωματωθεί πληροφορία που μπορεί να προϋπάρχει στην βάση και προχωράμε στην επόμενη ομάδα.

Αν, αντιθέτως, υπάρχουν και κανόνες με IDB κατηγορήματα στο σώμα προχωράμε στην προετοιμασία για να εισέλθουμε στο βρόχο υπολογισμού του σταθερού σημείου. Αρχικά δημιουργούνται οι δομές από τις οποίες θα παίρνουμε τους προσωρινούς κανόνες σε κάθε επανάληψη του βρόχου. Εδώ χρησιμοποιούνται δύο κλάσεις αντικειμένων, η TempRulesGenerator και η TempDbLookupRulesGenerator. Κάθε αντικείμενο της πρώτης κλάσης αναλαμβάνει να δημιουργεί με, τη μικρότερη δυνατή επιβάρυνση, τους προσωρινούς κανόνες που αντιστοιχούν σε έναν κανόνα της εισόδου. Αν ο υπολογισμός της απάντησης γίνεται μέσω μαγικής μεταγραφής και υπάρχουν κανόνες για τον έλεγχο ύπαρξης δεδομένων που αφορούν IDB κατηγορήματα στη βάση θα δημιουργηθεί μόνο ένα αντικείμενο από τη δεύτερη κλάση που θα παράγει όλους τους κατάλληλους, για κάθε βήμα, κανόνες αυτού του τύπου. Υπάρχουν δύο τρόποι υλοποίησης του βρόχου υπολογισμού του σταθερού σημείου του αλγορίθμου αποτίμησης datalog που παρουσιάστηκε στο προηγούμενο κεφάλαιο. Ο ένας είναι αυτός που χρησιμοποιήθηκε κατά την παρουσίαση του αλγορίθμου, με τα ονόματα των προσωρινών κατηγορημάτων να περιλαμβάνουν τον αριθμό της επανάληψης. Ο δεύτερος είναι με την διαδοχική εναλλαγή δύο διαφορετικών ειδών ονομάτων. Οι λεπτομέρειες φαίνονται στον παρακάτω πίνακα:

**Πίνακας 5: Διαφορετικοί τρόποι υλοποίησης βρόχου.**

| Επανάληψη | Πρώτος τρόπος | Δεύτερος τρόπος |
|---|---|---|
| 1η | $S_1 \leftarrow S_0 \cup \Delta_1$<br>temp2 ← …..<br>$\Delta_2 \leftarrow$ temp2 – $S_1$ | SA ← SB U ΔA<br>tempB ← …..<br>ΔB ← tempB – SA |
| 2η | $S_2 \leftarrow S_1 \cup \Delta_2$<br>temp3 ← …..<br>$\Delta_3 \leftarrow$ temp3 – $S_2$ | SB ← SA U ΔB<br>tempA ← …..<br>ΔA ← tempA – SB |

| | | |
|---|---|---|
| 3η | $S_3 \leftarrow S_2 \cup \Delta_3$<br>temp4 ← …..<br>$\Delta_4 \leftarrow temp4 - S_3$ | SA ← SB U ΔA<br>tempB ← …..<br>ΔB ← tempB – SA |
| 4η | $S_4 \leftarrow S_3 \cup \Delta_4$<br>temp5 ← …..<br>$\Delta_5 \leftarrow temp5 - S_4$ | SB ← SA U ΔB<br>tempA ← …..<br>ΔA ← tempA – SB |

Με τον δεύτερο τρόπο ανακυκλώνονται τα ίδια ονόματα προσωρινών κατηγορημάτων κάθε δεύτερη επανάληψη του βρόχου. Όπως γίνεται αντιληπτό με αυτό τον τρόπο αποφεύγεται ο εκ νέου υπολογισμός των προσωρινών κανόνων σε κάθε επανάληψη. Πράγματι, όταν έχει επιλεγεί ο δεύτερος τρόπος, οι δύο κλάσεις που παράγουν τέτοιου είδους κανόνες υπολογίζουν την κάθε ομάδα κανόνων μόνο την πρώτη φορά και την αποθηκεύουν για τις επόμενες φορές που θα ζητηθεί. Αυτό έχει ως συνέπεια τα κέρδη από την παράλληλη δημιουργία των προσωρινών κανόνων μειώνονται δραματικά. Στην πράξη αποδεικνύεται ότι το όποιο κέρδος αποκομίζουμε κατά τη δημιουργία τους δεν επαρκεί για να καλυφθεί το κόστος που επιφέρει η διαχείριση των επιπλέον νημάτων. Έτσι το χαρακτηριστικό αυτό απενεργοποιείται όταν χρησιμοποιείται αυτό το είδος προσωρινών κανόνων. Το μειονέκτημα της επαναχρησιμοποίησης των ίδιων ονομάτων είναι ότι πρέπει να εκδίδονται επιπλέον εντολές προς τη βάση για τον μηδενισμό τους ανάμεσα στις επαναλήψεις. Εντολές μηδενισμού απαιτούνται μόνο για τα «temp» και «Δ», αφού στο «S» συγκεντρώνεται σταδιακά η πληροφορία που θα αποτελέσει το τελικό αποτέλεσμα. Από την πλευρά της βάσης υπάρχει το πλεονέκτημα εξοικονόμησης χώρου καθώς τα ενδιάμεσα αποτελέσματα διαγράφονται αμέσως μόλις χρησιμοποιηθούν. Τα υπέρ και τα κατά κάθε τρόπου επηρεάζουν σε διαφορετικό βαθμό τα συστήματα βάσεων δεδομένων. Πιο συγκεκριμένα παρατηρήθηκε ότι όταν χρησιμοποιείται ο εξυπηρετητής RDF Fuseki, όπου επηρεάζει έντονα ο αριθμός των εντολών που πρέπει να εκτελεστούν, πετυχαίνει καλύτερες επιδόσεις με τον πρώτο τρόπο ενώ με τον PostreSQL server γρηγορότερος είναι ο δεύτερος τρόπος. Το σχήμα που επιλέχθηκε για τις βάσεις SQL συμβάλλει καθοριστικά αφού οι πληροφορίες για κάθε κατηγόρημα είναι συγκεντρωμένες σε έναν πίνακα. Ανάλογα με τον τύπο βάσης που θα χρησιμοποιηθεί επιλέγεται αυτόματα από το αντικείμενο Configurator και ο βέλτιστος τρόπος υλοποίησης του βρόχου.

Όπως αναφέρθηκε αν χρησιμοποιούνται διαφορετικά ονόματα για τα προσωρινά κατηγορήματα κάθε βήματος και υπάρχουν περισσότεροι από ένας διαθέσιμοι επεξεργαστές για την εκτέλεση της DatalogEvaluator οι προσωρινοί κανόνες θα δημιουργούνται παράλληλα σε νέα νήματα. Για να μεγιστοποιηθούν τα οφέλη από τον παράλληλο υπολογισμό τους αυτός πρέπει να ξεκινά από το σημείο με την μεγαλύτερη δυνατή «υπολογιστική απόσταση» από τις εντολές για την εκτέλεσή τους στη βάση. Έτσι οι

προσωρινοί κανόνες για την πρώτη επανάληψη ξεκινούν να δημιουργούνται έξω από τον βρόχο, προτού εκτελεστούν ακόμα οι πρώτοι κανόνες, αυτοί χωρίς κάποιο IDB κατηγόρημα στο σώμα. Εντός του βρόχου, πρώτα εκτελούνται οι εντολές ένωσης, κατόπιν οι προσωρινοί κανόνες και αμέσως μετά ξεκινά η προετοιμασία των προσωρινών κανόνων για την επόμενη επανάληψη. Μέχρι να έρθει η στιγμή της εκτέλεσής τους μεσολαβεί ολόκληρος ο βρόχος, καθώς και ο έλεγχος για τον τερματισμό του, δίνοντας τον μέγιστο δυνατό χρόνο στους ελεύθερους επεξεργαστές να ολοκληρώσουν τον υπολογισμό των προσωρινών κανόνων για την επόμενη επανάληψη.

Μόλις φτάσουμε στο σταθερό σημείο, δηλαδή σταματήσει να παράγεται νέα πληροφορία για όλα τα κατηγορήματα IDB της ομάδας, βγαίνουμε από τον βρόχο. Τα ενδιάμεσα κατηγορήματα «S» περιέχουν πλέον την τελική πληροφορία για τα κατηγορήματα IDB. Θα αφαιρεθούν τα επιπλέον προθέματα όπως ο αριθμός της επανάληψης ή τα A και B για να μπορούν να εντοπιστούν οι τελικές πληροφορίες χωρίς να απαιτείται να γνωρίζουμε σε ποια επανάληψη φτάσαμε στο σταθερό σημείο για κάθε ομάδα κανόνων. Όταν ολοκληρωθεί η αποτίμηση όλων των ομάδων κανόνων θα απαντηθεί το ερώτημα που τυχόν έχει δοθεί και στη συνέχεια θα προχωρήσουμε στην τελική διαμόρφωση της βάσης. Σε κάθε περίπτωση στο τέλος θα διαγραφεί από τη βάση όποιο κατηγόρημα περιλαμβάνει το UPP στο όνομά του άρα πρέπει με τις κατάλληλες μετονομασίες να το αφαιρέσουμε από τα τελικά ονόματα των κατηγορημάτων που θα διατηρηθούν. Οι επιλογές είναι πέντε:

- Ενσωμάτωση του συνόλου της παραγόμενης πληροφορίας. Όταν φτάσουμε στο σταθερό σημείο για μια ομάδα κανόνων οι πληροφορίες για τα IDB κατηγορήματά που περιλαμβάνει αποθηκεύονται με τα τελικά τους ονόματα έτσι στο σημείο της τελικής διαμόρφωσης της βάσης δεν απαιτείται κάποια επιπλέον μετονομασία.
- Ενσωμάτωση μόνο των γεγονότων που συνοδεύουν τους κανόνες. Τα γεγονότα που περιλαμβάνει η είσοδος ενσωματώνονται στη βάση απευθείας με το τελικό τους όνομα. Όσον αφορά στις παραγόμενες πληροφορίες έχουν αποθηκευτεί στο σύνολό τους με το UPP να περιλαμβάνεται στο όνομα.
- Επαναφορά της βάσης στην αρχική της κατάσταση. Σε αυτή την περίπτωση πέρα από την διαγραφή των πεδίων με το UPP θα αφαιρεθούν και τα γεγονότα που προστέθηκαν στην αρχή.
- Πλήρης διαγραφή της βάσης. Εδώ δεν ελέγχεται το UPP καθώς διαγράφονται όλες ανεξαιρέτως οι πληροφορίες.
- Διατήρηση όλων των κατηγορημάτων. Αν κανείς επιθυμεί να παρακολουθήσει την πορεία εξαγωγής των συμπερασμάτων η βάση μπορεί να αφεθεί ως έχει με τα ενδιάμεσα κατηγορήματα παρόντα. Σε αυτή την περίπτωση δεν διαγράφεται τίποτα.

## *4.2 Επικοινωνία με βάσεις δεδομένων*

Ένας από τους στόχους κατά την υλοποίηση ήταν η ευελιξία στην υποστήριξη διαφορετικών συστημάτων αποθήκευσης δεδομένων. Για αυτό το σκοπό δημιουργήθηκε η διεπαφή DatabaseAccess. Σε αυτήν ορίζονται οι απαραίτητες λειτουργίες που πρέπει να υποστηρίζει ένα υποσύστημα διαχείρισης βάσης δεδομένων για να μπορεί να χρησιμοποιηθεί για την αποτίμηση της μεταγραφής. Με αυτόν τον τρόπο παρέχεται η μέγιστη δυνατή διευκόλυνση για τη δημιουργία μελλοντικών επεκτάσεων του προγράμματος με σκοπό την υποστήριξη περισσότερων βάσεων δεδομένων, ακόμα και ριζικά διαφορετικής τεχνολογίας.

Οι μέθοδοι που περιλαμβάνει είναι οι εξής:

- **executeRule**: Εκτελεί έναν κανόνα στη βάση. Οφείλει να συνδυάσει τα δεδομένα από τα κατηγορήματα του σώματος λαμβάνοντας υπόψη τους περιορισμούς που αυτό επιβάλλει και να διοχετεύσει το αποτέλεσμα στην δομή που αναπαριστά το κατηγόρημα της κεφαλής.
- **insertFact**: Εισάγει στη βάση τα δεδομένα που περιέχει το γεγονός. Το γεγονός είναι η στοιχειώδης μονάδα πληροφορίας, αντιστοιχεί σε μία τριάδα RDF ή μία γραμμή πίνακα σε βάση SQL.
- **executeQuery**: Αποτιμά το ερώτημα πάνω στα δεδομένα της βάσης και επιστρέφει την απάντηση. Μοιάζει αρκετά με την εκτέλεση κανόνα με την διαφορά ότι οι πληροφορίες δεν κατευθύνονται σε κάποια δομή εντός της βάσης. Επιπλέον το άτομο στην κεφαλή του ερωτήματος μπορεί να έχει απεριόριστο αριθμό όρων καθώς αυτοί αντιπροσωπεύουν τις μεταβλητές για τις οποίες ενδιαφερόμαστε να αντλήσουμε πληροφορίες.
- **areEmpty**: Ελέγχει αν περιέχονται δεδομένα σε ένα σύνολο από κατηγορήματα. Χρησιμοποιείται για να ελεγχθεί αν έχουμε φτάσει στο σταθερό σημείο που σημαίνει το τέλος της παραγωγής νέων δεδομένων.
- **union**: Υπολογίζει το σύνολο της ένωσης των πληροφοριών δύο κατηγορημάτων και αποθηκεύει το αποτέλεσμα σε τρίτο κατηγόρημα. Φυσικά για να είναι δυνατή αυτή η πράξη τα δύο αρχικά κατηγορήματα πρέπει να έχουν τον ίδιο αριθμό όρων.
- **rename**: Αλλάζει το όνομα της δομής που αναπαριστά ένα κατηγόρημα στη βάση.
- **subtract**: Αφαιρεί από το κατηγόρημα που παίζει το ρόλο του αφαιρετέου την πληροφορία που είναι κοινή σε αυτό και το κατηγόρημα αφαιρέτη και αποθηκεύει το αποτέλεσμα σε ένα τρίτο κατηγόρημα. Και εδώ όλα τα κατηγορήματα πρέπει να έχουν τον ίδιο αριθμό όρων.

- **clenupDB**: Διαγράφει από τη βάση ότι περιέχει τη συμβολοσειρά που επισημαίνει τις προσωρινές πληροφορίες (Unique Processing Prefix - UPP) σε πεδίο όπου αποθηκεύονται ονόματα κατηγορημάτων. Οι πληροφορίες που θέλουμε να ενσωματώσουμε στη βάση μετά το πέρας της εκτέλεσης πρέπει να έχουν ήδη μετονομαστεί.
- **concatExistingInfo**: Χρησιμοποιείται για να ενσωματώσει προϋπάρχουσες πληροφορίες στη βάση για τα κατηγορήματα IDB. Αντιγράφει ότι πληροφορία υπάρχει για κάθε ένα από αυτά κάτω από τα νέα όνοματά τους που πλέον έχουν και το πρόθεμα που αντιστοιχεί στο κατάλληλο βήμα της διαδικασίας υπολογισμού τους.
- **delete**: Αυτή η μέθοδος διαγράφει μεμονωμένα γεγονότα από τη βάση.
- **isAlreadyIn**: Ελέγχει την ύπαρξη ενός γεγονότος στη βάση.
- **nullifyDB**: Διαγράφει όλα τα περιεχόμενα της βάσης.
- **getTotalRulesIssued**: Επιστρέφει τον συνολικό αριθμό των εντολών που έχουν δοθεί στον διαχειριστή της βάσης. Η λειτουργία αυτή είναι προαιρετική.
- **executeTranslatedRule**: Εκτελεί ήδη μεταφρασμένο κανόνα στη βάση. Σαν είσοδο δέχεται μία συμβολοσειρά με τις απαραίτητες εντολές στην κατάλληλη γλώσσα για άμεση εκτέλεση στη βάση. Πρόκειται για άλλη μια προαιρετική λειτουργία.
- **getPredicateNames**: Επιστρέφει όποιο όνομα υπάρχει στα πεδία όπου αποθηκεύονται ονόματα κατηγορημάτων. Χρησιμοποιείται για την κατασκευή και τον έλεγχο του UPP.
- **flush**: Καλείται μία φορά στο τέλος για να ενημερώσει τον διαχειριστή της βάσης ότι η τελευταία εντολή έχει δοθεί. Του δίνει την ευκαιρία να ολοκληρώσει τυχόν εκκρεμείς εργασίες αφού ολοκληρωθεί η εργασία με την βάση. Παραδείγματα εκκρεμών εργασιών είναι η εκτέλεση της τελευταίας ομάδας εντολών ενημέρωσης και η απενεργοποίηση νημάτων που έχουν δημιουργηθεί από τον χειριστή της βάσης.

## *4.3 Βασεις RDF*

Η υλοποίηση του οράματος του σημασιολογικού ιστού απαιτεί να καταστούν οι πόροι που είναι διαθέσιμοι στον παγκόσμιο ιστό κατανοητοί από τα τεχνικά μέσα τα οποία σήμερα χρησιμοποιούνται για λειτουργίες χαμηλότερου επιπέδου. Ένας μηχανισμός για να επιτευχθεί αυτό είναι μέσω της σημασιολογικής σήμανσης κάθε πόρου. Η σήμανση θα είναι κατανοητή από τα προγράμματα που θα λειτουργούν αυτόνομα στο περιβάλλον του σημασιολογικού ιστού. Για αυτό το σκοπό το W3C ανέπτυξε το Resource Description Framework (RDF,

πλαίσιο περιγραφής πόρων αν θέλουμε μια απόδοση στα ελληνικά). Η πιο πρόσφατη έκδοση του προτύπου RDF είναι η 1.1 και εκδόθηκε το 2014 [107]. Η έννοια πόρος είναι αρκετά γενική ώστε να περιλαμβάνει οτιδήποτε για το οποίο μπορεί να θελήσουμε να δώσουμε επιπλέον πληροφορίες. Έτσι, σε αντίθεση με τα δεδομένα RDF που περιλαμβάνουν τις πληροφορίες, οι ίδιοι οι πόροι τους οποίους αυτές αφορούν δεν είναι υποχρεωτικό να είναι προσβάσιμοι μέσω του ιστού. Στο πρότυπο RDF ορίζεται ότι οι αναφορές στους πόρους γίνονται μέσω των Internationalised Resource Identifiers (IRI). Αυτό είναι το πιο γενικό είδος αναγνωριστικών που χρησιμοποιούνται. Αποτελούν την εξέλιξη των Uniform Resource Identifiers (URI) και τα επεκτείνουν με τη δυνατότητα χρήσης χαρακτήρων Unicode. Κάθε URI, άρα και URL (Uniform Resource Locator), αποτελεί ένα έγκυρο IRI αλλά το αντίθετο δεν ισχύει πάντα. Κάθε πόρος καθορίζεται μοναδικά από ένα IRI και σε οτιδήποτε αντιστοιχεί ένα IRI είναι πόρος. Συχνά το IRI ενός πόρου αποτελεί έγκυρο URL το οποίο οδηγεί στη διεύθυνση που είτε βρίσκεται ο ίδιος ο πόρος είτε κάποια σελίδα που τον αναπαριστά ή δίνει επιπλέον διευκρινήσεις για αυτόν.

Τα δεδομένα RDF καταγράφονται υπό τη μορφή τριάδων {s, p, o} που περιλαμβάνουν κατά σειρά το υποκείμενο (subject, s), την ιδιότητα (property/predicate, p) και το αντικείμενο (object, o). Ο πόρος στον οποίο αναφέρεται η τριάδα είναι στη θέση του υποκείμενου. Η ιδιότητα συνδέει το υποκείμενο με το αντικείμενο. Το λεξιλόγιο V της RDF αποτελείται από δύο ξένα σύνολα: το I που περιλαμβάνει τα αναγνωριστικά IRIs που είδαμε προηγουμένως και το L που περιλαμβάνει τα λεκτικά (literals). Τα λεκτικά μπορούν να είναι δύο ειδών. Τα απλά (plain literals) που είναι κοινές συμβολοσειρές και τα λεκτικά συγκεκριμένου τύπου (typed literals). Οι τύποι διευκρινίζουν τον τρόπο που πρέπει να ερμηνευτεί το λεκτικό, για παράδειγμα αν είναι δεκαδικός, Boolean κλπ. Ξένο από το V είναι το σύνολο B των κενών κόμβων (blank ή b nodes). Οι κόμβοι αυτοί δεν ταυτοποιούνται από παγκοσμίως μοναδικά αναγνωριστικά αλλά έχουν αναγνωριστικά τοπικού χαρακτήρα που δεν υπερβαίνουν την εμβέλεια του γράφου στον οποίο βρίσκονται και χρησιμοποιούνται σε σημεία όπου η εισαγωγή μιας νέας οντότητας δεν εξυπηρετεί. Τα τμήματα των τριάδων {s, p, o} μπορούν να δημιουργηθούν από τα παραπάνω στοιχεία ως εξής:

- $s \in I \cup B$
- $p \in I$
- $o \in I \cup B \cup L$

Δεδομένα γραμμένα σε RDF μπορούν να παρασταθούν και με έναν κατευθυνόμενο γράφο αν θεωρηθούν κόμβοι τα υποκείμενα και αντικείμενα και οι ακμές σχηματιστούν από τις ιδιότητες που τα συνδέουν. Αυτό το μοντέλο διακρίνεται πέρα από την απλότητά του και από μεγάλη ευελιξία αφού ενσωματώνει το σχήμα της βάσης στα ίδια τα δεδομένα. Για αυτό το λόγο μια βάση RDF είναι ιδανική για εφαρμογές όπου μπορεί να υπάρχουν συχνές αλλαγές στη δομή της ίδιας της βάσης ενώ, για τον ίδιο λόγο, ο συνδυασμός δεδομένων από

διαφορετικές πηγές είναι μια πολύ απλή διαδικασία. Σε συνδυασμό με την προτυποποίηση από έναν οργανισμό του βεληνεκούς του W3C δεν εκπλήσσει το γεγονός ότι το πρότυπο έχει γνωρίσει ευρεία αποδοχή και σε ένα εκτεταμένο σύνολο εφαρμογών που δεν συνδέονται με τον σημασιολογικό ιστό.

### *4.3.1 SPARQL*

Για την πλήρη αξιοποίηση αποθηκών δεδομένων απαιτείται μια γλώσσα ερωτημάτων. Για δεδομένα που ακολουθούν τη μορφή που παρουσιάστηκε προηγουμένως υπάρχει μια σειρά προτάσεων όπως οι RQL [108], RDQL [109] και SPARQL[29]. Θα χρησιμοποιήσουμε την τελευταία που επίσης αποτελεί πρόταση του W3C. Απολαμβάνει ευρύτατη υποστήριξη και παρόλο που άλλες γλώσσες δημιουργήθηκαν νωρίτερα, η SPARQL έχει γίνει η κύρια γλώσσα για την διαχείριση δεδομένων σε RDF. Το όνομα της είναι το αναδρομικό ακρώνυμο **S**PARQL **P**rotocol **A**nd **R**DF **Q**uery **L**anguage.

Ο εξυπηρετητής SPARQL που χρησιμοποιούμε εδώ είναι ο Fuseki[30]. Αποτελεί τμήμα του πακέτου ανάπτυξης εφαρμογών Jena του ιδρύματος Apache και είναι πρόγραμμα ανοιχτού κώδικα. Αυτή τη στιγμή είναι διαθέσιμη η δεύτερη κύρια έκδοσή του. Η επικοινωνία με τον Fuseki γίνεται μέσω του πρωτοκόλλου HTTP πράγμα που τον καθιστά ιδανικό σε περιπτώσεις όπου απαιτείται απομακρυσμένη πρόσβαση. Στη συνέχεια του κεφαλαίου παρουσιάζονται οι εντολές που χρησιμοποιούνται για τη διαχείριση της βάσης RDF και οι λειτουργίες στις οποίες αντιστοιχούν.

### *4.3.2 Εντολές SPARQL*

Στην επικοινωνία με τον fuseki θα χρησιμοποιηθούν δύο είδη εντολών. Τα ερωτήματα (queries) που ως στόχο έχουν την ανάσυρση πληροφοριών ήδη αποθηκευμένων στη βάση και οι εντολές ενημέρωσης (updates) που τροποποιούν τα περιεχόμενά της. Σε αυτό το σημείο θα

[29] https://www.w3.org/TR/2013/REC-sparql11-overview-20130321/

[30] https://jena.apache.org/documentation/fuseki2/

αναφερθούμε στις απολύτως απαραίτητες λειτουργίες, περισσότερες λεπτομέρειες υπάρχουν στο [110] για ερωτήματα και στο [111] για τις εντολές ενημέρωσης.

Κάθε άτομο αντιστοιχεί σε μία τριάδα RDF. Όπως αναφέρθηκε και προηγουμένως, στην πρώτη θέση μιας τριάδας αυτού του τύπου βρίσκουμε το υποκείμενο ακολουθούμενο την ιδιότητα και το αντικείμενο. Ένα κατηγόρημα ενός όρου όπως το father (X) αντιστοιχεί στην τριάδα { ?X a father }. Το «a» αποτελεί συντομογραφία του URIref[31] http://www.w3.org/1999/02/22-rdf-syntax-ns#type. Ένα URIref είναι ένα URI που στο τέλος, μετά το διαχωριστικό «#», περιλαμβάνει ένα αναγνωριστικό τμήματος (fragment identifier). Το συγκεκριμένο URIref αντιστοιχεί στον πόρο που περιγράφεται στο τμήμα «type» της σελίδας http://www.w3.org/1999/02/22-rdf-syntax-ns. Πρόκειται για αναγνωριστικό που αντιστοιχεί σε ισχύον URL. Αν το ακολουθήσουμε θα διαπιστώσουμε ότι η ιδιότητα αυτή έχει σαν πεδίο ορισμού το σύνολο των πόρων και σαν σύνολο τιμών έχει τις κλάσεις. Στην περιγραφή της ιδιότητας ορίζεται ότι μια τριάδα με αυτή την ιδιότητα σημαίνει ότι το υποκείμενο είναι στιγμιότυπο της κλάσης που αναφέρεται στην θέση του αντικειμένου. Ένα κατηγόρημα με δύο όρους π.χ. fatherOf (X, Y) αντιστοιχεί στην τριάδα { ?X fatherOf ?Y }. Το κατηγόρημα σε αυτή την περίπτωση αποτελεί την ιδιότητα. Το αντιμετωπίζουμε δηλαδή ως τον ρόλο «fatherOf» που συνδέει το υποκείμενο με το αντικείμενο.

Το όνομα ενός κατηγορήματος όπως το «fatherOf» δεν αποτελεί ένα έγκυρο IRI και δεν μπορεί να χρησιμοποιηθεί ως έχει καθώς είναι μια γραμματοσειρά χωρίς την απαραίτητη μορφή. Πρέπει είτε να συνδυαστεί με ένα βασικό URI για τον σχηματισμό ενός URIref ή να δίνεται ως έτοιμο, συμβατό αναγνωριστικό. Το τελευταίο είναι σύνηθες στην περίπτωση που το κατηγόρημα αποτελεί μέρος κάποιας οντολογίας OWL. Αν το κατηγόρημα δεν δίνεται σε μορφή έγκυρου αναγνωριστικού τότε προστίθεται ως αναγνωριστικό τμήματος σε ένα προεπιλεγμένο URI. Επειδή τα αναγνωριστικά που χρησιμοποιούνται στην RDF είναι συνήθως μεγάλα, στις εντολές SPARQL παρέχεται η δυνατότητα ορισμού προθεμάτων. Ένα παράδειγμα ορισμού προθέματος είναι το: «prefix rdf: http://www.w3.org/1999/02/22-rdf-syntax-ns#». Το «rdf» είναι η ετικέτα στην οποία θα αντιστοιχεί η συμβολοσειρά που ακολουθεί μετά το «:». Ορίζοντας το παραπάνω πρόθεμα μπορούμε αντί για το πλήρες URIref http://www.w3.org/1999/02/22-rdf-syntax-ns#type να γράψουμε rdf:type. Η ετικέτα επιτρέπεται να είναι κενή και στη συνέχεια θεωρούμε ότι η κενή ετικέτα έχει οριστεί να

[31] Κάθε URIref αποτελεί έγκυρο IRIref, όπως συμβαίνει με τα URIs/IRIs.

αντιστοιχεί στο προεπιλεγμένο URI που χρησιμοποιείται για να φέρει σε έγκυρη μορφή τα ονόματα των κατηγορημάτων[32]. Έτσι το «*:father*» αποτελεί έγκυρο αναγνωριστικό.

#### *4.3.2.1 Εισαγωγή γεγονότος*

Εδώ έχουμε την εισαγωγή μιας πλήρους τριάδας αφού λόγω της απουσίας σώματος δεν είναι δυνατό να υπάρχουν μεταβλητές. Όταν πρόκειται να εισαχθούν έτοιμες τριάδες στη βάση χρησιμοποιείται η εντολή INSERT DATA. Οι ακριβείς εντολές που αντιστοιχούν σε κατηγόρημα ενός ή δύο όρων φαίνονται στο παρακάτω παράδειγμα:

| | |
|---|---|
| father(nick). | *INSERT DATA {<nick> a :father.}* |
| fatherOf(nick, jim). | *INSERT DATA {<nick> :father <jim>.}* |

#### *4.3.2.2 Εκτέλεση πλήρους κανόνα*

Όταν υπάρχει σώμα, μπορούμε να έχουμε πέρα από σταθερές και μεταβλητές. Το σώμα του κανόνα, μαζί με σταθερές που μπορεί να βρίσκονται στην κεφαλή του θα επιβάλλουν τους περιορισμούς στους οποίους πρέπει να υπακούν τα δεδομένα που θα εισαχθούν στη βάση κάτω από το όνομα του κατηγορήματος της κεφαλής. Έστω ότι έχουμε να εκτελέσουμε τον κανόνα headPr(X, constA) ← bodyPr1(X, Y), bodyPr2(Y, constB). Η εντολή προς τον διαχειριστή της βάσης RDF που αντιστοιχεί σε αυτόν είναι:

```
INSERT   {  ?X   :headPr    <constA>  }
WHERE    {  ?X   :bodyPr1   ?Y.
            ?Y   :bodyPr2   <constB>.  }
```

[32] Ένας ακόμη λόγος που επιλέχθηκε η κενή ετικέτα, πέρα από την απλότητα στο κείμενο των εντολών, είναι ότι η επικοινωνία μέσω HTTP ωφελείται από τον περιορισμό του μήκους των συμβολοσειρών που κωδικοποιούν τις εντολές προς τον Fuseki.

Όπως βλέπουμε κάθε άτομο μετατρέπεται σε μια τριάδα με το κατηγόρημα να αποτελεί την ιδιότητα ή το αντικείμενο ανάλογα με τον αριθμό των όρων που αντιστοιχούν σε αυτό. Το αν κάποιος όρος είναι μεταβλητή ή σταθερά δεν επηρεάζει τη δομή της τριάδας. Το μόνο που αλλάζει είναι ότι ενώ οι σταθερές περιβάλλονται ανάμεσα σε "<" και ">", οι μεταβλητές υποδηλώνονται με ένα αγγλικό ερωτηματικό πριν το όνομά τους.

#### *4.3.2.3 Εκτέλεση ερωτήματος*

Το ερώτημα είναι ένας κανόνας με ένα ιδιαίτερο κατηγόρημα στην κεφαλή. Έχει όνομα «q» ή «query» και μπορεί να έχει απεριόριστο αριθμό όρων διότι οι όροι αυτοί αποτελούν τις μεταβλητές για τις οποίες ενδιαφερόμαστε να αντλήσουμε πληροφορία. Όλες οι μεταβλητές της κεφαλής περιέχονται στο σώμα το οποίο επιβάλλει τους επιθυμητούς περιορισμούς. Έστω το ερώτημα query(X, Y, Z) ← pr1(X, const), pr2(X, Y), pr3(Z). Το ερώτημα σε γλώσσα SPARQL που αντιστοιχεί σε αυτό είναι:

```
SELECT      ?X    ?Y    ?Z
WHERE   {   ?X    :pr1  <const>.
            ?X    :pr2   ?Y.
            ?Z     a    :pr3.  }
```

Το *SELECT* ακολουθείται από τις μεταβλητές που περιέχονται στην κεφαλή του ερωτήματος και για τις οποίες θα επιστραφούν δεδομένα ενώ οι περιορισμοί του σώματος κωδικοποιούνται στο *WHERE*.

#### *4.3.2.4 Ένωση*

Αυτή η εντολή χρησιμοποιείται για να υπολογίσουμε την ένωση των πληροφοριών δύο κατηγορημάτων. Το αποτέλεσμα θα αποθηκευτεί σε ένα τρίτο κατηγόρημα. Για να το πετύχουμε αυτό θα χρησιμοποιήσουμε τον τελεστή *UNION*. Ο τελεστής αυτός επιστρέφει την ένωση των αποτελεσμάτων τα οποία επιλέγονται από τους περιορισμούς που αποτελούν τα ορίσματά του. Όλα τα κατηγορήματα θα πρέπει να έχουν τον ίδιο αριθμό όρων. Έστω ότι

θέλουμε να υπολογίσουμε την ένωση των κατηγορημάτων με έναν όρο: *resPr = pr1 U pr2*. Η εντολή RDF που θα την υλοποιήσει είναι η:

```
INSERT      { ?s   a   :resPr }
WHERE    { { ?s   a   :pr1   }
                UNION
             { ?s   a   :pr2   } }
```

Με το *UNION* αντιστοιχίζονται στην μεταβλητή «s» όλες οι πληροφορίες που βρίσκονται στη βάση σαν πρώτος όρος των κατηγορημάτων *pr1* και *pr2*. Στη συνέχεια εισάγονται μέσω του *INSERT* σαν τιμές του μοναδικού όρου του κατηγορήματος *resPr*.

#### *4.3.2.5 Έλεγχος για κενά κατηγορήματα*

Για να ελέγξουμε αν περιέχεται στη βάση πληροφορία για κάποιο κατηγόρημα μπορούμε να χρησιμοποιήσουμε το ερώτημα:

```
SELECT      ?x
WHERE       { ?x   :pred   ?y }
LIMIT       1
```

Επιλέγεται η μεταβλητή «x» η οποία αντιστοιχεί στον πρώτο όρο του κατηγορήματος *pred*. Το *LIMIT 1* στο τέλος του ερωτήματος περιορίζει το επιστρεφόμενο αποτέλεσμα (και την υπολογιστική προσπάθεια που αυτό απαιτεί) στην μία γραμμή αφού το μόνο που ενδιαφέρει είναι αν περιέχεται κάποια πληροφορία για το κατηγόρημα *pred*. Στην περίπτωση που μας ενδιαφέρει αν περιέχεται κάποια πληροφορία σε μια ομάδα κατηγορημάτων μπορούμε να χρησιμοποιήσουμε τον τελεστή *UNION* που είδαμε προηγουμένως. Το ερώτημα θα είναι:

```
SELECT      ?x
WHERE    { { ?x  :pred1  ?y  }
               UNION
            { ?x   a    :pred2 }
               UNION
                 ...
```

```
                UNION
            { ?x :predN ?y } }
LIMIT       1
```

Εντάσσοντας στην αλυσίδα ενώσεων στο *WHERE* τμήμα της εντολής τριάδες κατάλληλης μορφής ανάλογα με το πόσους όρους περιέχει κάθε κατηγόρημα και αντιστοιχίζοντας την μεταβλητή «*x*» στον πρώτο όρο κάθε τέτοιας τριάδας τότε, με τη βοήθεια του *UNION*, το αποτέλεσμα θα είναι να επιστραφούν όλοι οι πρώτοι όροι των αναφερόμενων κατηγορημάτων. Το *LIMIT 1* στο τέλος θα περιορίσει το επιστρεφόμενο αποτέλεσμα στην μια και μοναδική γραμμή που είναι απαραίτητη για να διαπιστώσουμε την ύπαρξη πληροφορίας για κάποιο από όλα τα κατηγορήματα.

#### *4.3.2.6 Μετονομασία*

Για την αλλαγή του ονόματος κάποιου κατηγορήματος στη βάση απαιτείται η τροποποίηση της πληροφορίας που βρίσκεται αποθηκευμένη στη θέση της ιδιότητας ή του αντικειμένου, ανάλογα με το αν πρόκειται για κατηγόρημα ενός ή δύο όρων. Αυτό δεν μπορεί να γίνει απευθείας αλλά θα πρέπει να δώσουμε μια "διπλή" εντολή η οποία θα διαγράφει τις τριάδες με το παλιό όνομα και θα προσθέτει νέες με το τροποποιημένο. Η εντολή αυτή για ένα κατηγόρημα ενός όρου είναι:

```
DELETE      { ?s   a   :oldPr  }
INSERT      { ?s   a   :newPr }
WHERE       { ?s   a   :oldPr  }
```

#### *4.3.2.7 Αφαίρεση*

Εδώ θέλουμε να υλοποιήσουμε τη λειτουργία της αφαίρεσης από το κατηγόρημα στο ρόλο του αφαιρετέου δεδομένα που είναι κοινά με το κατηγόρημα αφαιρέτη και να αποθηκεύσουμε το αποτέλεσμα σε ένα τρίτο κατηγόρημα. Θα χρειαστεί να χρησιμοποιήσουμε το φίλτρο "*NOT EXISTS*" το οποίο θα αποτρέψει την επιλογή δεδομένων που ικανοποιούν τον περιορισμό που το ακολουθεί. Έστω ότι επιθυμούμε να εφαρμοστεί η

πράξη αυτή στα παρακάτω κατηγορήματα ενός όρου: *resPr = pr1 – pr2*. Η εντολή που θα χρησιμοποιήσουμε είναι:

```
INSERT                { ?s   a   :resPr  }
WHERE                 { ?s   a   :pr1
FILTER NOT EXISTS     { ?s   a   :pr2  }  }
```

#### *4.3.2.8 Ενσωμάτωση πληροφορίας*

Για να ενσωματώσουμε την πληροφορία ενός κατηγορήματος σε ένα άλλο χρησιμοποιούμε την εντολή:

```
INSERT        { ?s   ?p   :newPr  }
WHERE         { ?s   ?p   :oldPr   }
```

#### *4.3.2.9 Διαγραφή βάσης*

Για να διαγράψουμε όλες τις τριάδες που περιέχει η βάση εκτελούμε την εντολή:

```
DELETE        { ?s ?p ?o }
WHERE         { ?s ?p ?o }
```

Όλες οι τριάδες ταιριάζουν στον περιορισμό *{ ?s ?p ?o }* οπότε διαγράφονται από το *DELETE*.

#### *4.3.2.10 Διαγραφή γεγονότος*

Ένα γεγονός αντιστοιχεί σε μία συγκεκριμένη τριάδα στη βάση. Για παράδειγμα το γεγονός *Father(nick)* αντιστοιχεί στην τριάδα: *{ <nick>  a  <Father> }*. Για τη διαγραφή του εκτελούμε την εντολή:

```
DELETE DATA         { <nick>   a   <Father>  }
```

#### *4.3.2.11 Έλεγχος για ύπαρξη γεγονότος*

Για να ελεγχθεί αν περιέχεται στη βάση μια συγκεκριμένη τριάδα χρησιμοποιείται η εντολή:

*ASK { <nick> a <Father> }*

#### *4.3.2.12 Λήψη ονομάτων κατηγορημάτων*

Πρέπει να ανασύρουμε από την βάση όλα τα ονόματα που περιέχονται σε πεδία όπου μπορεί να αποθηκευτεί όνομα κατηγορήματος. Το πρώτο από τα υποψήφια πεδία είναι αυτό της ιδιότητας. Εκεί, ως γνωστόν βρίσκονται τα ονόματα κατηγορημάτων με δύο όρους. Όνομα κατηγορήματος μπορεί να υπάρχει και στο πεδίο του αντικειμένου, σε τριάδες του τύπου { ?s a ?o }. Σε αυτή την περίπτωση πρόκειται για κατηγόρημα του ενός όρου. Η εντολή που χρησιμοποιείται και για τις δύο περιπτώσεις συνδυαστικά και είναι:

```
SELECT DISTINCT   ?p
WHERE             {  { ?s  ?p  ?o }
                     UNION
                     { ?d  a  ?p } }
```

Χρησιμοποιούμε την λέξη *DISTINCT* μετά το *SELECT* για να επιστραφεί μία μόνο φορά το κάθε όνομα.

#### *4.3.2.13 Καθαρισμός βάσης*

Με αυτή την εντολή επιθυμούμε να διαγράψουμε από την βάση τριάδες οι οποίες αντιστοιχούν σε κατηγορήματα τα οποία περιέχουν στο όνομά τους ένα πρόθεμα το οποίο δίνεται. Πρέπει να ελέγχονται κατηγορήματα είτε με έναν είτε με δύο όρους, τα οποία, όπως αναφέρθηκε προηγουμένως, αντιστοιχούν στο πεδίο της ιδιότητας ή του αντικειμένου σε τριάδες με το «a» σαν ιδιότητα. Αυτά ακριβώς είναι τα πεδία όπου οφείλουμε να αναζητήσουμε το πρόθεμα. Θα χρησιμοποιήσουμε τις δύο επόμενες εντολές για να καλύψουμε και τις δύο κατηγορίες:

*DELETE { ?s ?p ?o }*
*WHERE { ?s ?p ?o FILTER regex( str(?p), "UPP") }*

*DELETE { ?s a ?o }*
*WHERE { ?s a ?o FILTER regex( str(?o), "UPP") }*

Η συνάρτηση *regex(str(?p), "UPP")* επιστρέφει τις τριάδες που περιέχουν την συμβολοσειρά *UPP* στη θέση στην οποία αντιστοιχεί το "*?p*", στο συγκεκριμένο παράδειγμα πρόκειται για την θέση της ιδιότητας. Αυτές επιλέγονται από τον τελεστή *FILTER* για να εφαρμοστεί η εντολή *DELETE* που δηλώνεται στην πρώτη γραμμή.

### *4.3.3 Διαχειριστής βάσης RDF*

Οι απαραίτητες λειτουργίες για την υποστήριξη βάσης RDF συγκεντρώνονται στο πακέτο logicOverDB.dbAccess.RDF. Οι κλάσεις αυτού του πακέτου αναλαμβάνουν την σύνταξη των κατάλληλων εντολών στην γλώσσα SPARQL για την εκτέλεση των λειτουργιών που απαιτούνται και την επικοινωνία του προγράμματος με τον εξυπηρετητή της βάσης RDF, fuseki.

Η RDFDatabaseHandler αποτελεί την βασική κλάση στο πακέτο διαχείρισης βάσης RDF. Τα αντικείμενα αυτής της κλάσης υλοποιούν την διεπαφή DatabaseAccess μέσω της οποίας γίνεται η επικοινωνία του υπόλοιπου κώδικα με τη βάση δεδομένων.

Ο κατασκευαστής του RDFDatabaseHandler δέχεται σαν παράμετρο ένα αντικείμενο τύπου Configurator από το οποίο αντλεί τις παρακάτω πληροφορίες:

- Την διεύθυνση του dataset του SPARQL server στο οποίο θα εργαστούμε (datasetURL). Είναι της μορφής: *ServerURL:Port/Dataset name/*. Δεν περιλαμβάνει τα τελικά "update" ή "query" τα οποία διακρίνουν την διεύθυνση που δέχεται εντολές ενημέρωσης και ερωτήματα. Η διάκριση των διευθύνσεων γίνεται αυτόματα ανάλογα με τον τύπο της εντολής που μεταβιβάζουμε στον RDF server. Προσοχή θέλει το γεγονός ότι στο όνομα του dataset γίνεται διάκριση μεταξύ πεζών και κεφαλαίων γραμμάτων.
- Αν θα γίνεται σε ξεχωριστό νήμα η τροφοδοσία του RDF server (threadedFusekiHandling).

- Αν θα γίνεται παράλληλα η μετάφραση των κανόνων σε εντολές RDF (generateUpdatesFromDLRulesConcurrently).
- Πληροφορίες για την καταγραφή δεδομένων σχετικά με την πορεία του υπολογισμού όπως το επίπεδο της λεπτομέρειας των μηνυμάτων που θα τυπώνονται στην πρότυπη έξοδο (verbosity), το αντικείμενο για την γενική καταγραφή δεδομένων από όλα τα υποσυστήματα και το αν και σε ποιο αρχείο θα πρέπει να καταγραφούν οι εντολές που μεταβιβάζονται στον RDF server.

Στη συνέχεια αρχικοποιεί, με τα απαραίτητα προθέματα, το πρώτο αντικείμενο τύπου UpdateRequest. Στα αντικείμενα αυτού του τύπου φορτώνονται οι εντολές ενημέρωσης της βάσης RDF. Έτσι όταν συγκεντρωθούν οι απαιτούμενες εντολές αποστέλλονται σαν ομάδα στον εξυπηρετητή για εκτέλεση. Τέλος προβαίνει στις απαραίτητες ενέργειες για να προετοιμάσει την παράλληλη εκτέλεση όπου αυτό απαιτείται. Πιο συγκεκριμένα αν έχει ζητηθεί να τροφοδοτούνται παράλληλα οι εντολές στον server τότε εκκινεί το νήμα με τον κατάλληλο κώδικα. Επιπλέον μειώνει στο μισό το μέγιστο χώρο που μπορεί να καταλαμβάνει κάθε UpdateRequest, αφού πλέον θα πρέπει να μπορούν να διατηρούνται δύο τέτοια πακέτα ταυτόχρονα στην κύρια μνήμη. Αν πρέπει να γίνεται παράλληλα η μετάφραση των κανόνων τότε εκκινεί όσα νήματα μπορούν να υποστηριχθούν από τους διαθέσιμους πόρους. Τέλος ανασύρονται τα ονόματα όλων των κατηγορημάτων που υπάρχουν στη βάση. Θα διατηρούνται στη μνήμη και θα ενημερώνονται ανάλογα με τις εντολές που εκτελούνται. Αξιοποιούνται για την απόρριψη εντολών όταν αναφέρονται σε κατηγορήματα που δεν υπάρχουν στη βάση. Σε αντίθεση με την σχεσιακή βάση που θα δούμε στη συνέχεια, στην RDF δεν είναι απαραίτητο να προϋπάρχει κάποια υποδομή για να είμαστε σε θέση να αποθηκεύσουμε πληροφορίες για νέα κατηγορήματα. Όμως η εξοικονόμηση πόρων που επιτυγχάνεται στις περιπτώσεις όπου τα γεγονότα στην βάση αφορούν ένα μικρό υποσύνολο των κατηγορημάτων των κανόνων είναι θεαματική πράγμα που επιβάλλει την υλοποίηση του χαρακτηριστικού αυτού και στην RDFDatabaseHandler.

Οι εντολές προς την βάση διαχωρίζονται σε δύο γενικές κατηγορίες, τα ερωτήματα και τις εντολές ενημέρωσης. Ο τρόπος διαχείρισής τους διαφοροποιείται ανάλογα με την κατηγορία οπότε θα παρουσιαστούν σε δύο ενότητες.

### *4.3.4 Εντολές ενημέρωσης*

Οι εντολές ενημέρωσης τροποποιούν το περιεχόμενο της βάσης ενημερώνοντας το αποθηκευμένο μοντέλο με τα νέα στοιχεία που προκύπτουν από την αποτίμηση του προγράμματος datalog. Σε αυτή την κατηγορία βρίσκουμε τις εντολές για την εκτέλεση κανόνων, την ένωση και αφαίρεση πληροφοριών κατηγορημάτων, την μετονομασία ή την διαγραφή κάποιου κατηγορήματος, την επιλεκτική εκκαθάριση και την ολική διαγραφή της βάσης και τέλος την εισαγωγή ή αφαίρεση της πληροφορίας που αντιστοιχεί σε κάποιο συγκεκριμένο γεγονός.

Ο τρόπος εκτέλεσης των εντολών αυτών είναι ο εξής: Αρχικά συντίθεται, σε γλώσσα SPARQL, η εντολή που αντιστοιχεί στην λειτουργία που πρέπει να εκτελεστεί. Η εντολή εισάγεται σε ένα αντικείμενο τύπου UpdateRequest. Στη συνέχεια αξιοποιείται το αντικείμενο αυτό για την αποστολή της ομάδας εντολών που περιέχει στον fuseki server ο οποίος με τη σειρά του πραγματοποιεί στη βάση τις αλλαγές που υπαγορεύονται. Επειδή η επικοινωνία με τον fuseki γίνεται μέσω του πρωτοκόλλου HTTP υπάρχουν ορισμένοι περιορισμοί. Ο σημαντικότερος έγκειται στον αριθμό των συνδέσεων που διατίθενται από το λειτουργικό σύστημα. Για την αποστολή κάθε UpdateRequest δημιουργείται μία νέα σύνδεση η οποία δεσμεύει μία θύρα. Για να ολοκληρωθεί η αποτίμηση μιας πολύπλοκης μεταγραφής με μεγάλο αριθμό κανόνων μπορεί να απαιτηθούν ακόμα και δεκάδες χιλιάδες εντολές προς βάση. Αν αυτές αποστέλλονται μεμονωμένα τότε πολύ γρήγορα οδηγούμαστε σε εξάντληση των διαθέσιμων από το λειτουργικό σύστημα θυρών. Ο δεύτερος περιορισμός έχει να κάνει με την επιβάρυνση που υπάρχει για την αποστολή ενός UpdateRequest. Για να γίνει αυτό απαιτείται η εκκίνηση μιας νέας σύνδεσης προς τον server πράγμα που έχει κόστος και σε χρόνο, πέρα από τους άλλους πόρους που δεσμεύει. Για αυτόν τον λόγο είναι απαραίτητο να συνδυαστούν όσο το δυνατόν περισσότερες εντολές σε κάθε UpdateRequest προτού αυτό σταλεί στον server. Υπάρχουν και εδώ περιορισμοί, ο αριθμός των εντολών που μπορούν να σταλούν ταυτόχρονα δεν μπορεί να μεγαλώσει απεριόριστα για δύο λόγους. Κατ' αρχάς ο ίδιος ο αλγόριθμος απαιτεί σε ορισμένα σημεία την αποστολή ερωτήματος στη βάση. Στο τέλος κάθε κύκλου εκτέλεσης προσωρινών κανόνων πρέπει να ελεγχθεί αν είχαμε παραγωγή νέας γνώσης, πράγμα που επιτυγχάνεται με την αποστολή ερωτήματος το οποίο δεν μπορεί να περιληφθεί στο UpdateRequest και αποστέλλεται χωριστά. Έτσι μόνο οι εντολές που αντιστοιχούν σε λειτουργίες του ίδιου κύκλου μπορούν να συνδυαστούν. Ένας ακόμα περιορισμός υπάρχει σε περιπτώσεις όπου έχουμε πολλούς και πολύπλοκους κανόνες, με μεγάλο αριθμό κατηγορημάτων IDB που λειτουργούν πολλαπλασιαστικά στο πλήθος των προσωρινών κανόνων. Δεν είναι απίθανο σε αυτές τις συνθήκες να απαιτηθεί η εκτέλεση χιλιάδων εντολών σε κάθε κύκλο. Η ταυτόχρονη διατήρηση στην κεντρική μνήμη μίας τόσο

εκτεταμένης ομάδας εντολών μπορεί να μην είναι δυνατή, ειδικά αν δεν έχει προβλεφθεί η παραχώρηση επιπλέον μνήμης.

Όλα τα παραπάνω καταδεικνύουν την σημασία που έχει η σωστή ομαδοποίηση των εντολών ενημέρωσης της βάσης. Ο μηχανισμός που ελέγχει την ομαδοποίηση έχει ως σημείο εκκίνησης την μέθοδο executionLine στην οποία καταλήγουν όλες οι έτοιμες εντολές ενημέρωσης και από τις δέκα μεθόδους που τις παράγουν. Η μέθοδος αυτή προσθέτει στο UpdateRequest την εντολή που λαμβάνει κάθε φορά που καλείται και ταυτόχρονα ελέγχει αν το μέγεθος του συνόλου των έτοιμων εντολών είναι τέτοιο που να επιβάλλει την άμεση αποστολή τους στον server για εκτέλεση.

Η προώθηση του πακέτου των έτοιμων εντολών προς τον server γίνεται από την μέθοδο executeUpdates. Πρώτα πρέπει να διασφαλίσει ότι το UpdateRequest πράγματι περιέχει εντολές διαφορετικά θα επιβαρυνθούμε, όπως αναφέρθηκε παραπάνω, για την αποστολή ενός κενού UpdateRequest. Αυτός ο έλεγχος είναι απαραίτητος αφού η συγκεκριμένη μέθοδος δεν καλείται μόνο από την executionLine, οπότε και είναι βέβαιο ότι υπάρχουν εντολές προς εκτέλεση. Στη συνέχεια αν έχουμε εκχωρήσει την επικοινωνία με τον server σε άλλο νήμα το UpdateRequest μεταβιβάζεται σε αυτό, διαφορετικά εκτελείται άμεσα. Τέλος το UpdateRequest ανανεώνεται με ένα νέο αντικείμενο της ίδιας κλάσης στο οποίο προστίθενται τα απαραίτητα προθέματα ενώ το παλιό αφήνεται να διαγραφεί από την μνήμη.

### *4.3.5 Παράλληλη μετάφραση κανόνων*

Υπάρχει η δυνατότητα για παράλληλη μετάφραση των προς εκτέλεση κανόνων σε γλώσσα SPARQL η οποία διαφοροποιεί ελαφρώς την παραπάνω διαδικασία. Αντί να μεταφραστεί απευθείας ο κανόνας και να κληθεί η μέθοδος executionLine για να προωθήσει την εντολή δημιουργείται ένα νέο αντικείμενο τύπου RDFRuleExecutionString με τον προς μετάφραση κανόνα σαν παράμετρο. Τα αντικείμενα αυτού του τύπου περιέχουν τον απαραίτητο κώδικα για την μετάφραση των κανόνων και υλοποιούν την διεπαφή Callable για να μπορούν να χρησιμοποιηθούν σε ξεχωριστά νήματα. Μια νέα μέθοδος διατηρεί τις απαραίτητες αναφορές στα μελλοντικά αποτελέσματα του παράλληλου υπολογισμού της μετάφρασης και ελέγχει αν το μέγεθος του UpdateRequest επιβάλλει την εκτέλεση των εντολών. Επιπλέον, στις μεθόδους executeUpdates και executionLine της προηγούμενης παραγράφου προστίθεται ένας έλεγχος για την ύπαρξη κανόνων που μεταφράζονται παράλληλα. Στην executeUpdates ο έλεγχος αυτός διασφαλίζει ότι δεν θα προωθηθεί προς τον διαχειριστή της βάσης ένα UpdateRequest πριν ληφθεί το αποτέλεσμα της παράλληλης μετάφρασης με συνέπεια να

παραλειφθεί να εκτελεστεί κάποιος κανόνας. Ο έλεγχος στην executionLine απαιτείται για την περίπτωση που έρθει κάποια εντολή που δεν αντιστοιχεί σε κανόνα. Η εντολή αυτή θα μεταφραστεί αμέσως και η executionLine θα κληθεί να την προσθέσει στο UpdateRequest. Αν υπάρχουν κανόνες που μεταφράζονται παράλληλα θα πρέπει οι εντολές που αντιστοιχούν σε αυτούς να προστεθούν πριν από την νέα εντολή. Για παράδειγμα μία εντολή αφαίρεσης, με την οποία υπολογίζονται τα νέα δεδομένα που παρήχθησαν για κάποιο κατηγόρημα στον τελευταίο κύκλο εκτέλεσης προσωρινών κανόνων, πρέπει να εκτελεστεί αφού εκτελεστούν όλοι οι προσωρινοί κανόνες που αντιστοιχούν σε αυτό το κατηγόρημα. Όπως γίνεται αντιληπτό αν η προσθήκη των εντολών από κανόνες που μεταφράζονται παράλληλα αφεθεί να γίνει όταν ολοκληρωθεί ο παράλληλος υπολογισμός υπάρχει η πιθανότητα να επηρεαστεί η ορθότητα του προγράμματος.

### *4.3.6 Ερωτήματα*

Οι μέθοδοι που παράγουν ερωτήματα είναι τέσσερεις. Οι executeQuery, areEmpty, doesPhraseExistInAName και η isAlreadyIn. Οι πρώτες τρεις αιτούνται δεδομένα από τη βάση πράγμα που υλοποιείται από ερωτήματα τύπου SELECT ενώ η τέταρτη αναζητά την ύπαρξη κάποιας συγκεκριμένης τριάδας στη βάση και χρησιμοποιεί ερωτήματα τύπου ASK τα οποία επιστρέφουν απλώς ένα ναι ή όχι. Ο τρόπος υποβολής των ερωτημάτων των δύο τύπων διαφέρει μόνο στην μέθοδο που χρησιμοποιείται στο τελευταίο βήμα οπότε θα παρουσιαστούν μαζί. Όπως και για τις εντολές ενημέρωσης, για να υποβληθεί ένα ερώτημα στη βάση και να επιστραφούν τα αποτελέσματα απαιτείται μία σύνδεση HTTP. Μόλις ολοκληρωθεί και η επεξεργασία των αποτελεσμάτων στέλνεται η εντολή που σημαίνει το τέλος της διαδικασίας εκτέλεσης του ερωτήματος και έτσι απελευθερώνονται οι πόροι που είχαν δεσμευτεί για τον σκοπό αυτό από την εικονική μηχανή της Java. Όμως η σύνδεση που χρησιμοποιήθηκε για την αποστολή του ερωτήματος και των αποτελεσμάτων δεν απελευθερώνεται άμεσα. Λόγω του σχεδιασμού του πρωτοκόλλου TCP[33] όταν και τα δύο μέρη συμφωνήσουν να κλείσει η σύνδεση, αυτή παραμένει ενεργή στην κατάσταση “TIME_WAIT”. Θα παραμείνει σε αυτή την κατάσταση για χρόνο ίσο με το διπλάσιο του μέγιστου χρόνου ζωής (MLS) των πακέτων του πρωτοκόλλου. Έτσι εξασφαλίζεται ότι πακέτα τα οποία έχουν καθυστερήσει να παραδοθούν δεν θα επηρεάσουν κάποια επόμενη σύνδεση που χρησιμοποιεί την θύρα που μόλις ελευθερώθηκε αφού αυτά θα έχουν

[33] https://tools.ietf.org/html/rfc793

απορριφθεί προτού δημιουργηθεί η νέα σύνδεση. Τα περισσότερα σύγχρονα λειτουργικά συστήματα επιτρέπουν την χρήση της θύρας πριν την λήξη αυτού του χρονικού περιθωρίου αν ο συνδυασμός διεύθυνσης/θύρας του άλλου άκρου είναι διαφορετικός, αφού τυχόν αργοπορημένα πακέτα μπορούν να διαχωριστούν και να απορριφθούν. Δυστυχώς στην περίπτωσή μας αυτό δεν μπορεί να γίνει αφού η επικοινωνία γίνεται με τον ίδιο server. Επομένως πρέπει να ληφθούν μέτρα για την αντιμετώπιση της πιθανότητας εξάντλησης των διαθέσιμων θυρών. Το πρώτο είναι προληπτικό αφορά στην εξάλειψη της βασικότερης πηγής πολλών συνεχόμενων ερωτημάτων. Η πηγή αυτή δεν ήταν άλλη από τον περιοδικό έλεγχο για το αν μια ομάδα κατηγορημάτων περιέχει δεδομένα. Αντί να χρησιμοποιείται ένα ερώτημα για κάθε κατηγόρημα μέχρι να βρεθεί κάποιο με δεδομένα ή να αποδειχθούν όλα άδεια χρησιμοποιείται ένα συνδυαστικό ερώτημα όπως παρουσιάστηκε στην προηγούμενη ενότητα. Με αυτόν τον τρόπο κερδίζουμε αρκετά και σε χρόνο αφού η δημιουργία της σύνδεσης εισάγει μια μη αμελητέα καθυστέρηση. Αν παρόλα αυτά φτάσουμε στο σημείο να εξαντληθούν οι θύρες, τότε θα πρέπει να περιμένουμε το οριστικό κλείσιμο των παλαιών συνδέσεων. Υπεύθυνος για την εξάντληση των διαθέσιμων θυρών μπορεί να είναι, πέρα από τον ρυθμό με τον οποίο παράγονται ερωτήματα, και ο επικοινωνιακός φόρτος από τρίτες εφαρμογές. Αν η μέθοδος που αναλαμβάνει την εκτέλεση των ερωτημάτων αντιληφθεί ότι η εκτέλεση κάποιου ερωτήματος απέτυχε λόγω έλλειψης διαθέσιμων συνδέσεων, θα προσπαθήσει εκ νέου να το εκτελέσει μετά από ένα χρονικό διάστημα. Εξαιτίας της απρόβλεπτης φύσης του προβλήματος αυτού ο χρόνος αναμονής δεν είναι σταθερός αλλά προσαρμόζεται ανάλογα με τις συνθήκες που αντιμετωπίζουμε κάθε φορά. Πιο συγκεκριμένα ο χρόνος διπλασιάζεται ανάμεσα σε δύο συνεχόμενες αποτυχημένες προσπάθειες εκτέλεσης ενός ερωτήματος ενώ αντίθετα υποτετραπλασιάζεται την πρώτη φορά που μπαίνουμε σε αυτή την διαδικασία για ένα νέο ερώτημα. Έτσι δίνεται η δυνατότητα προσαρμογής του προς τα κάτω αν ο επικοινωνιακός φόρτος κάποια στιγμή μειωθεί.

Τέλος πρέπει να σημειωθεί ότι πριν από την εκτέλεση ερωτήματος πρέπει να βεβαιωθούμε ότι δεν υπάρχει κάποια ομάδα εντολών που αναμένει να σταλεί στον server. Αν υπάρχει τότε στέλνεται άμεσα. Αν επιπλέον είναι ενεργοποιημένη η παράλληλη εκτέλεση των εντολών ενημέρωσης και ζητηθεί να εκτελεστεί κάποιο ερώτημα ενόσω αυτή βρίσκεται σε εξέλιξη τότε θα πρέπει να περιμένουμε την ολοκλήρωσή της προτού συνεχίσουμε. Διαφορετικά υπάρχει ο κίνδυνος η εξυπηρέτηση του ερωτήματος να προηγηθεί της εκτέλεσης κάποιων εντολών με αποτέλεσμα να λάβουμε λανθασμένη απάντηση.

### *4.3.7 Παράλληλη ενημέρωση βάσης*

Για τις περιπτώσεις όπου έχουμε στη διάθεσή μας παραπάνω από έναν λογικό επεξεργαστή ή ο server βρίσκεται σε διαφορετικό μηχάνημα επιτυγχάνεται καλύτερη απόδοση αν η εκτέλεση των εντολών ενημέρωσης της βάσης γίνεται παράλληλα από ένα νέο νήμα. Έτσι είναι δυνατή η συνέχιση των υπολογισμών όσο ο server εκτελεί προηγούμενες εντολές. Την παράλληλη εκτέλεση των εντολών ενημέρωσης αναλαμβάνει η κλάση FusekiUpdater. Η κύρια μέθοδος που τρέχει στο νέο νήμα, παράλληλα με το υπόλοιπο πρόγραμμα, περιλαμβάνει έναν βρόχο που επαναλαμβάνεται μέχρι να δοθεί το σήμα ότι ολοκληρώθηκε η αποστολή εντολών. Στο πρώτο μέρος του βρόχου υπάρχει ο έλεγχος για την ύπαρξη εντολών προς εκτέλεση. Αν δεν βρεθούν εντολές τότε θα τεθεί σε κατάσταση αναμονής μέχρι να ειδοποιηθεί από το κύριο νήμα. Τότε θα ελέγξει εκ νέου αν υπάρχουν εντολές. Εδώ ο έλεγχος επαναλαμβάνεται διότι υπάρχει περίπτωση να έχουμε τυχαίες ειδοποιήσεις που δεν προέρχονται από την αναμενόμενη πηγή. Θα προχωρήσει στο δεύτερο μέρος του βρόχου αν έρθουν νέες εντολές ή αν δοθεί το σήμα για τον τερματισμό. Εκεί θα γίνει η εκτέλεση των εντολών και η απόρριψη του χρησιμοποιημένου UpdateRequest. Αν δεν έχει δοθεί το σήμα του τερματισμού τότε επιστρέφει στο πρώτο μέρος του βρόχου αναμένοντας νέο πακέτο εντολών. Τρεις είναι οι δημόσιες μέθοδοι με τις οποίες επικοινωνεί το υπόλοιπο πρόγραμμα με τον FusekiUpdater. Οι μέθοδοι αυτές, όπως και η κύρια μέθοδος που αναφέρθηκε προηγουμένως, είναι συγχρονισμένες για να αποφεύγονται προβλήματα στην παράλληλη επικοινωνία μεταξύ των δύο νημάτων. Αυτό σημαίνει ότι μια από αυτές μπορεί να εκτελείται κάθε στιγμή. Αν τη στιγμή που κάποια από τις μεθόδους εκτελείται ένα άλλο νήμα θελήσει και αυτό να χρησιμοποιήσει κάποια μέθοδο του FusekiUpdater τότε αναγκαστικά θα πρέπει να περιμένει είτε να ολοκληρωθεί αυτή που βρίσκεται υπό εκτέλεση είτε να μπει σε κατάσταση αναμονής. Έτσι επιτυγχάνεται ο σκοπός που δεν είναι άλλος από την αποτροπή της ταυτόχρονης εκτέλεσης και τροποποίησης κάποιας ομάδας εντολών, πράγμα που θα είχε απρόβλεπτες συνέπειες.

Τα UpdateRequests μεταβιβάζονται από τον RDFDatabaseHandler στον FusekiUpdater μέσω της μεθόδου setUpdate. Αρχικά ελέγχεται αν υπάρχει ήδη προηγούμενο UpdateRequest που εκτελείται και αν αυτό ισχύει το νήμα που εκτελεί τον RDFDatabaseHandler μπαίνει σε κατάσταση αναμονής. Όταν ολοκληρωθεί η εκτέλεση αυτού θα ειδοποιηθεί και θα προχωρήσει στην μεταβίβαση του νέου UpdateRequest και στη συνέχεια θα ειδοποιήσει με τη σειρά της το νήμα που εκτελεί τις ενημερώσεις της βάσης να ξεκινήσει την επεξεργασία του νέου UpdateRequest. Δεν υλοποιήθηκε η δυνατότητα να διατηρούνται περισσότερα UpdateRequests διότι η μέγιστη αποδοτικότητα επιτυγχάνεται με την αποστολή όσο το δυνατόν μεγαλύτερων ομάδων εντολών, αφού μειώνεται ο αριθμός των συνδέσεων που

απαιτούνται για την αποτίμηση της μεταγραφής. Μοιράζοντας την διαθέσιμη μνήμη σε δύο πακέτα εντολών επιτυγχάνεται ο καλύτερος δυνατός συμβιβασμός μεταξύ του μεγέθους των ομάδων και της αξιοποίησης της δυνατότητας για παράλληλη δημιουργία και εκτέλεση των εντολών ενημέρωσης. Η δεύτερη δημόσια μέθοδος είναι η waitUntilUpdatesExecuted. Καλείται όταν πρέπει να βεβαιωθούμε ότι έχουν εκτελεστεί όλες οι εντολές ενημέρωσης, όπως για παράδειγμα πριν την υποβολή κάποιου ερωτήματος. Η λειτουργία της είναι παρόμοια με την setUpdate. Μπλοκάρει αν εντοπίσει ότι υπάρχει κάποια ομάδα εντολών και αναμένει το πέρας της εκτέλεσής της. Σε αντίθεση με την setUpdate δεν προσθέτει νέες εντολές ενημέρωσης.

Η μέθοδος signalTheEnd χρησιμοποιείται για να σημάνει το τέλος της διαδικασίας. Καλείται μια φορά μετά την φόρτωση του τελευταίου UpdateRequest, ενημερώνει την αντίστοιχη σημαία και ειδοποιεί το νήμα που εκτελεί τις εντολές στη βάση να επεξεργαστεί, αν δεν το έχει κάνει ήδη, το τελευταίο UpdateRequest και να κλείσει.

## *4.4 Βάσεις SQL*

Η αποτίμηση της μεταγραφής μπορεί να γίνει και πάνω σε σχεσιακή βάση δεδομένων με τη χρήση της γλώσσας SQL (Structured Query Language). Η συγκεκριμένη τεχνολογία είναι ευρέως διαδεδομένη και υπάρχει πληθώρα συστημάτων διαχείρισης βάσεων αυτού του τύπου. Η παρούσα υλοποίηση έχει δοκιμαστεί με το σύστημα PostgeSQL[34], ένα ανοιχτού κώδικα σύστημα διαχείρισης σχεσιακών βάσεων δεδομένων με αντικειμενοστραφή χαρακτηριστικά (Object Relational Database Management System). Στη συνέχεια παρουσιάζονται οι ακριβείς εντολές που χρησιμοποιούνται. Όλες οι εντολές διαχείρισης της βάσης SQL που θα δούμε εμπίπτουν στο πρότυπο ANSI SQL (& ISO/IEC). Αναμένεται να λειτουργούν χωρίς τροποποιήσεις και σε άλλα συστήματα βάσεων αυτού του τύπου. Δεν υπάρχει απόλυτη βεβαιότητα διότι, παρά την ύπαρξη προτυποποίησης από τα τέλη της δεκαετίας του '80, υπάρχουν αρκετές διαφορές μεταξύ των διαφόρων υλοποιήσεων. Για να επιτευχθεί η αποτίμηση σε διαφορετικά συστήματα διαχείρισης βάσεων που υποστηρίζουν την SQL ίσως απαιτηθούν ορισμένες περιορισμένης έκτασης και κατά κύριο λόγο συντακτικού χαρακτήρα τροποποιήσεις.

[34] https://www.postgresql.org

### *4.4.1 Δομή της βάσης δεδομένων*

Στην παρεχόμενη υλοποίηση, οι πληροφορίες στη βάση SQL αποθηκεύονται σε πίνακες που έχουν το ίδιο όνομα με το κατηγόρημα στο οποίο αντιστοιχούν και αριθμό στηλών ανάλογο του πλήθους των όρων αυτού. Αν έχουμε έναν όρο δημιουργείται μία στήλη με το όνομα "term" ενώ αν έχουμε δύο οι στήλες ονομάζονται "term1" και "term2". Ο τύπος δεδομένων για κάθε στήλη είναι συμβολοσειρά μεταβλητού μήκους, με μέγιστο μήκος τους 255 χαρακτήρες. Σαν πρωτεύον κλειδί ορίζεται το σύνολο των στηλών κάθε πίνακα για να εξασφαλιστεί ότι κάθε γεγονός θα αποθηκευτεί στη βάση μια φορά. Το συγκεκριμένο σχήμα διατηρεί συγκεντρωμένες τις πληροφορίες για κάθε κατηγόρημα με αποτέλεσμα την αποδοτική υλοποίηση του ελέγχου για ύπαρξη πληροφορίας και της ολικής διαγραφής τους.

### *4.4.2 Εντολές SQL*

Οι εντολές διαφοροποιούνται ανάλογα με τον αριθμό των όρων που περιέχουν τα άτομα στα οποία αναφέρονται. Για παράδειγμα αν έχουμε άτομο με έναν όρο, όπως το pred(trm), στο WHERE τμήμα της εντολής θα γράψουμε: p.term = 'trm'. Για ένα άτομο με δύο όρους όπως το pred(trmA, trmB) η έκφραση σε αυτό το σημείο θα γίνει: p.term1 = 'termA' AND p.term2 = 'termB'. Για την απλοποίηση της παρουσίασης οι εντολές που θα δοθούν παρακάτω θα αναφέρονται κυρίως σε άτομα με έναν μόνο όρο.

Παρατηρήσεις:

- Τα διπλά εισαγωγικά καταδεικνύουν αναγνωριστικά ονόματα για στοιχεία της βάσης, όπως πινάκες ή στήλες. Τα ονόματα των κατηγορημάτων πρέπει να είναι εντός διπλών εισαγωγικών για να διατηρείται η διάκριση μεταξύ κεφαλαίων και πεζών γραμμάτων στα ονόματα των πινάκων της βάσης στα οποία αντιστοιχούν. Για τα ονόματα των στηλών τα διπλά εισαγωγικά παραλείπονται διότι χρησιμοποιούνται τα ονόματα term, term1 και term2 που δεν περιέχουν κεφαλαία γράμματα.
- Τα μονά εισαγωγικά συμβολίζουν μία σταθερή συμβολοσειρά. Οι σταθερές που περιέχονται ως όροι στα άτομα των κανόνων θα περικλείονται σε μονά εισαγωγικά όταν χρησιμοποιούνται σε εντολές SQL.

#### *4.4.2.1 Εισαγωγή γεγονότος*

Εδώ γίνεται απλή εισαγωγή δεδομένων στη βάση. Έστω ότι το γεγονός το οποίο θέλουμε να εισαχθεί στη βάση είναι το *headPr(const)* ←. Το αποτέλεσμα της εκτέλεσης του στη βάση είναι η δημιουργία μιας γραμμής στον πίνακα που αντιστοιχεί στο *headPr* που περιέχει τις σταθερές που περιλαμβάνονται στους όρους. Η εντολή που χρησιμοποιείται είναι:

```
INSERT INTO        "headPr"
SELECT             'const'
WHERE NOT EXISTS
   (   SELECT      *
       FROM        "headPr" p
       WHERE       p.term='const'    )
```

Η πρώτη γραμμή δείχνει που θα διοχετευθούν οι νέες πληροφορίες. Η δεύτερη γραμμή δείχνει τι ακριβώς θα επιλεγεί για εισαγωγή και στις εντολές που εισάγουν γεγονότα θα περιλαμβάνει τις σταθερές του ατόμου της κεφαλής. Το *WHERE NOT EXISTS* στο τέλος της εντολής διασφαλίζει ότι προστίθενται στον πίνακα μόνο σειρές που δεν βρίσκονται ήδη σε αυτόν. Η παρένθεση περιλαμβάνει μία πλήρη εντολή SQL. Αυτή η εντολή επιλέγει τις γραμμές του πίνακα *headPr* με περιεχόμενα ακριβώς ίδια με αυτά που προκύπτουν από το γεγονός που επιθυμούμε να εισάγουμε. Στην περίπτωση που υπάρχει στον πίνακα *headPr* μια τέτοια σειρά το *NOT EXISTS* που προηγείται της φωλιασμένης εντολής θα παραβιαστεί και δεν θα γίνει η εισαγωγή. Αν αφαιρεθεί αυτός ο έλεγχος και κάποια εντολή έχει ως αποτέλεσμα να προστεθεί σειρά που υπάρχει ήδη τότε θα παραβιαστεί ο περιορισμός του μοναδικού κλειδιού και ο εξυπηρετητής SQL δεν θα εκτελέσει την εντολή επιστρέφοντας μήνυμα λάθους. Ο αστερίσκος μετά το *SELECT* υποδεικνύει την επιλογή όλων των σειρών του πίνακα που ακολουθεί το *FROM* στην επόμενη γραμμή. Στο *FROM* της φωλιασμένης εντολής το αναγνωριστικό του πίνακα ακολουθείται από το γράμμα *p*. Αυτή η φράση είναι συντακτικά ισοδύναμη με την «*"headPr" AS p»* η οποία αποδίδει στο αναγνωριστικό "*headPr*" το προσωρινό όνομα *p* το οποίο ισχύει για το υπόλοιπο της εντολής. Η συγκεκριμένη δυνατότητα θα αξιοποιείται όπου είναι δυνατό διότι με αυτό τον τρόπο περιορίζουμε το μήκος της εντολής ενώ παράλληλα απλοποιούμε τη διαδικασία δημιουργίας της περιορίζοντας της επιμέρους συμβολοσειρές οι οποίες την συνθέτουν.

#### *4.4.2.2 Εκτέλεση κανόνα*

Η εκτέλεση του κανόνα έχει ως αποτέλεσμα την εισαγωγή σειρών που επαληθεύουν τους περιορισμούς του σώματος στον πίνακα που αντιστοιχεί στο κατηγόρημα της κεφαλής. Το είδος των εντολών που χρησιμοποιούνται θα παρουσιαστεί μέσω ενός παραδείγματος. Έστω ότι ο προς εκτέλεση κανόνας είναι ο *headPr(X, constA) ←bodyPr1(X, Y), bodyPr2(Y, constB).* Η εντολή ενημέρωσης της βάσης που αντιστοιχεί σε αυτόν είναι:

```
INSERT INTO         "headPr"
SELECT DISTINCT     "bodyPr1".term1, 'constA'
FROM                "bodyPr1", "bodyPr2"
WHERE
   (    "bodyPr1".term2="bodyPr2".term1  AND  "bodyPr1".term2='constB'    )
   AND NOT EXISTS
   (    SELECT      *
        FROM        "headPr" p
        WHERE       p.term1="bodyPr1".term1  AND  p.term2='constA'    )
```

Η εντολή διαφοροποιείται σε σχέση με την απλή εισαγωγή γεγονότος από την δεύτερη γραμμή. Έχει προστεθεί η λέξη *DISTINCT* που διασφαλίζει ότι δεν θα επιχειρηθεί να γίνει εισαγωγή στον πίνακα *headPr* της ίδιας σειράς δύο φορές, πράγμα που θα διέκοπτε την εκτέλεση του κανόνα λόγω της παραβίασης του περιορισμού του μοναδικού κλειδιού. Αυτό μπορεί να συμβεί, ακόμα και αν δεν υπάρχουν διπλές εγγραφές στους πίνακες της βάσης, από την ένωση (join) που υπονοείται στην τρίτη γραμμή. Διαφορετικοί συνδυασμοί των δεδομένων στους πίνακες είναι δυνατό να οδηγήσουν στο ίδιο αποτέλεσμα. Το *FROM* της τρίτης γραμμής ακολουθείται από τα ονόματα όλων των κατηγορημάτων του σώματος και υποδεικνύει στον εξυπηρετητή SQL ότι οι πίνακες που αντιστοιχούν σε αυτά θα πρέπει να συνδυαστούν. Στο *WHERE* που ακολουθεί κωδικοποιούνται οι περιορισμοί που επιβάλλει το σώμα του κανόνα. Αυτή η γραμμή υποδεικνύει στον εξυπηρετητή SQL ποιες πληροφορίες πρέπει να διατηρηθούν από τον συνδυασμό των πινάκων. Υπάρχουν δύο είδη περιορισμών. Ο πρώτος προκύπτει από πολλαπλές εμφανίσεις της ίδιας μεταβλητής στο ίδιο ή σε διαφορετικό άτομο. Ο περιορισμός αυτός έχει την εξής μορφή: "όνομα κατηγορήματος 1ης εμφάνισης".όνομα στήλης = "όνομα κατηγορήματος 2ης εμφάνισης".όνομα στήλης. Έτσι διασφαλίζεται ότι θα επιλεγούν στοιχεία της ένωσης που έχουν την ίδια τιμή στα πεδία τα οποία αντιστοιχούν σε όρους όπου εμφανίζεται η ίδια μεταβλητή. Ο δεύτερος περιορισμός προκαλείται από την ύπαρξη σταθερών και έχει την μορφή: "όνομα κατηγορήματος με

σταθερά".όνομα στήλης = 'σταθερά'. Στο τέλος, όπως και στην περίπτωση των γεγονότων, υπάρχει ο περιορισμός *NOT EXISTS* για να εισαχθούν μόνο νέες σειρές στον πίνακα headPr.

Στην περίπτωση που κάποιο κατηγόρημα εμφανίζεται περισσότερες από μία φορές στο σώμα στις επιπλέον εμφανίσεις του θα αναφερόμαστε μέσω ψευδωνύμου όπως φαίνεται στο επόμενο παράδειγμα για τον κανόνα *least(X, Y) ← least(X,Z), least(Z, Y).*

```
INSERT INTO          "least"
SELECT DISTINCT      "least".term1, "least2".term2
FROM                 "least", "least" "least2"
WHERE
   (    "least".term1=" least".term2       )
   AND NOT EXISTS
   (    SELECT       *
        FROM         "least" p
        WHERE        p.term1="least".term1  AND  p.term2="least2".term2    )
```

### *4.4.2.3 Υπό συνθήκη δημιουργία πίνακα*

Αυτή η εντολή δημιουργεί ένα νέο πίνακα στην περίπτωση που αυτός δεν υπάρχει ήδη στη βάση. Έστω ότι επιθυμούμε να δημιουργήσουμε πίνακα για το κατηγόρημα *pred* το οποίο εμφανίζεται σε άτομα με έναν όρο. Η εντολή SQL είναι:

```
CREATE TABLE IF NOT EXISTS        "pred"
        (   term VARCHAR(255) NOT NULL,
            PRIMARY KEY (term)   );
```

Η παρένθεση που ακολουθεί την πρώτη γραμμή περιγράφει τα ονόματα και τους τύπους των στηλών του πίνακα όπως και τους περιορισμούς οι οποίοι ισχύουν για αυτόν. Στην συγκεκριμένη περίπτωση στην δεύτερη γραμμή ορίζεται η στήλη *term* η οποία είναι τύπου *VARCHAR(225)* δηλαδή περιέχει συμβολοσειρές μεταβλητού μήκους με μέγιστο μήκος 225 χαρακτήρες. Το *NOT NULL* που ακολουθεί απαγορεύει την ύπαρξη σειρών στις οποίες αυτή η στήλη θα είναι κενή. Η τρίτη γραμμή εισάγει τον περιορισμό *PRIMARY KEY* ο οποίος αναφέρει ότι το πρωτεύον κλειδί του πίνακα είναι η στήλη *term*. Αν είχαμε κατηγόρημα δύο όρων τότε και οι δύο στήλες του θα περιλαμβάνονταν στο πρωτεύον κλειδί.

#### *4.4.2.4 Εκτέλεση ερωτήματος*

Το ερώτημα, όπως αναφέρθηκε και σε προηγούμενη ενότητα, είναι ένας κανόνας με ένα ειδικό κατηγόρημα στην κεφαλή με όνομα «q» ή «query». Μπορεί να έχει απεριόριστο αριθμό όρων καθώς οι όροι αυτοί αποτελούν τις μεταβλητές για τις οποίες ενδιαφερόμαστε να αντλήσουμε πληροφορία και υποχρεωτικά περιέχονται στο σώμα το οποίο επιβάλλει τους επιθυμητούς περιορισμούς. Έστω το ερώτημα *query(X, Y, Z) ← pr1(X, const), pr2(X, Y), pr3(Z)*. Το ερώτημα SQL που θα αποσταλεί στον εξυπηρετητή είναι:

```
SELECT DISTINCT   "pr1".term1, "pr2".term1, "pr2".term2
FROM              "pr1", "pr2", "pr3"
WHERE        (    "pr1".term1="pr2".term1, "pr1".term2='const',
                  "pr2".term2="pr3".term        )
```

Η πρώτη γραμμή περιλαμβάνει τις στήλες που θα επιλεγούν για να συνθέσουν την απάντηση, οι οποίες αντιστοιχούν στις μεταβλητές της κεφαλής του ερωτήματος. Πιο συγκεκριμένα, επιλέγεται η στήλη που αντιστοιχεί στην πρώτη εμφάνιση της μεταβλητής στο σώμα[35]. Η δεύτερη γραμμή περιλαμβάνει τους πίνακες που πρέπει να συνδυαστούν και είναι αυτοί που αντιπροσωπεύουν τα κατηγορήματα του σώματος. Η τρίτη είναι προαιρετική και περιλαμβάνει τους περιορισμούς που μπορεί να επιβάλλει το σώμα του ερωτήματος στις τιμές των μεταβλητών. Περιορισμοί επιβάλλονται από το σώμα στην περίπτωση που υπάρχει σταθερά ή μεταβλητές που εμφανίζονται περισσότερες από μία φορές. Στις περιπτώσεις όπου έχουμε περισσότερες από μία εμφανίσεις ενός κατηγορήματος στο σώμα του ερωτήματος ακολουθείται η ίδια διαδικασία που ισχύει και για την εκτέλεση κανόνων.

#### *4.4.2.5 Έλεγχος για ύπαρξη γεγονότος*

Για να ελέγξουμε την ύπαρξη κάποιου γεγονότος π.χ. *pred('const')* ← στη βάση θα εκτελέσουμε το παρακάτω ερώτημα SQL:

[35] Αν κάποια μεταβλητή της κεφαλής δεν εμφανίζεται στο σώμα τότε υπάρχει συντακτικό λάθος στο ερώτημα, το οποίο θα έχει ανιχνευτεί προτού φτάσουμε στο στάδιο της εκτέλεσής του στη βάση.

*SELECT*
*FROM “pred”*
*WHERE term='const'*

Η πρώτη γραμμή αυτού του ερωτήματος ίσως ξενίσει. Εδώ το αναμενόμενο “*” έχει απομακρυνθεί. Το κενό μετά το SELECT έχει ως αποτέλεσμα να επιστραφεί μια απάντηση χωρίς καμία στήλη. Με αυτό τον τρόπο απαλλάσσουμε τον server από την υποχρέωση να επιστρέψει κάποια τιμή αποθηκευμένη στον πίνακα για το κατηγόρημα *pred*, συνήθως έκτασης μερικών δεκάδων χαρακτήρων, η οποία και θα μας ήταν αδιάφορη. Ο έλεγχος για την ύπαρξη γραμμής στην απάντηση είναι αρκετός για να βεβαιωθούμε για την ύπαρξη του γεγονότος στη βάση.

Υπάρχουν και σε άλλες εντολές σημεία στα οποία δεν μας ενδιαφέρει τι θα επιστραφεί από το *SELECT,* να όπου δεν το κάνουμε αυτό. Όλα βρίσκονται σε υποερωτήματα αμέσως μετά από το “EXISTS”. Επειδή αυτές οι περιπτώσεις είναι πολύ κοινές υπάρχει πρόβλεψη στο πρότυπο της SQL, ήδη από την έκδοση του '92, το “*” μετά το SELECT να αντιστοιχεί σε μία οποιαδήποτε τιμή.

#### *4.4.2.6 Έλεγχος για κενό κατηγόρημα*

Για να ελέγξουμε αν η βάση περιέχει πληροφορία για κάποιο κατηγόρημα ελέγχουμε αρχικά αν υπάρχει πίνακας για αυτό. Στην περίπτωση που υπάρχει πρέπει να ελεγχθεί αν είναι κενός. Αυτό γίνεται με το παρακάτω ερώτημα SQL:

*SELECT*
*FROM “predicate”*
*LIMIT 1*

Αυτό το ερώτημα, λόγω του *LIMIT 1* στο τέλος, θα ανασύρει από τη βάση το πολύ μία σειρά από τον πίνακα *predicate*. Φυσικά η απάντηση είναι καταφατική αν δεν υπάρχει ο πίνακας ή το ερώτημα δεν επιστρέψει καμία σειρά.

#### *4.4.2.7 Ένωση*

Η ένωση των πληροφοριών δύο κατηγορημάτων σε ένα τρίτο (*resPr = pred1 U pred2*) γίνεται σε τρία βήματα. Πρώτα δημιουργείται, αν δεν υπάρχει ήδη, ο πίνακας που θα αντιπροσωπεύει το *resPr*. Κατόπιν εισάγονται στο *resPr* οι πληροφορίες του *pred1* και στη συνέχεια προστίθενται οι πληροφορίες του *pred2*. Αυτό επιτυγχάνεται με την εκτέλεση της παρακάτω εντολής πρώτα για το *pred1* και μετά για το *pred2*.

```
INSERT INTO   "resPr"
SELECT        *
FROM          "pred1" AS o
WHERE NOT EXISTS
    (   SELECT        *
        FROM          "resPr" AS r
        WHERE         r.term = o.term     );
```

Το «*AS o*» στο τέλος της τρίτης γραμμής προσδίδει στο αναγνωριστικό που προηγείται, εδώ το *pred1*, το σύντομο προσωρινό όνομα «o» με το οποίο θα αναφερόμαστε σε αυτό για το υπόλοιπο της εντολής SQL. Το τελευταίο κομμάτι της εντολής εξασφαλίζει ότι δεν θα εισαχθούν σειρές που υπάρχουν ήδη στον πίνακα *resPr*. Φυσικά για να είναι δυνατή η ένωση τα δύο κατηγορήματα πρέπει να έχουν τον ίδιο αριθμό όρων.

#### *4.4.2.8 Μετονομασία*

Αυτές τις εντολές τις χρησιμοποιούμε όταν χρειάζεται να αλλάξουμε το όνομα ενός πίνακα.

```
ALTER TABLE IF EXISTS "oldName" RENAME TO "newName";
ALTER INDEX IF EXISTS "oldName_pkey" RENAME TO "newName_pkey";
```

Η πρώτη αλλάζει, αν υπάρχει ο πίνακας *oldName*, το όνομα από *oldName* σε *newName*. Η δεύτερη μετονομάζει τον περιορισμό πρωτεύοντος κλειδιού *oldName_pkey* σε *newName_pkey* επίσης αν αυτός υπάρχει. Η πρώτη εντολή για μετονομασία του πίνακα δεν επηρεάζει το όνομα του περιορισμού για το πρωτεύον κλειδί. Αν το αφήσουμε με το παλιό

όνομα και επιχειρηθεί να δημιουργηθεί εκ νέου πίνακας με το παλιό όνομα, στην προσθήκη του περιορισμού για αυτόν θα παρουσιαστεί σφάλμα καθώς θα υπάρχει ήδη περιορισμός με το ίδιο όνομα στη βάση. Παρότι οι δύο περιορισμοί αναφέρονται σε διαφορετικούς πίνακες, δεν μπορούν να συνυπάρξουν με το ίδιο όνομα. Όπως γίνεται αντιληπτό, για να διατηρηθεί η τάξη στη βάση είναι απαραίτητη και η δεύτερη εντολή για τη μετονομασία του περιορισμού κάθε φορά που αλλάζει όνομα ένας πίνακας.

#### *4.4.2.9 Αφαίρεση*

Σκοπός αυτής της εντολής είναι να εκτελέσει την πράξη *resPr = pr1 – pr2*. Από τις πληροφορίες που υπάρχουν για το *pr1* αφαιρούνται όσες είναι κοινές με το *pr2* και το αποτέλεσμα αποθηκεύεται στο *resPr*.

```
INSERT INTO   "resPr"
SELECT        *
FROM          "pr1" AS o1
WHERE NOT EXISTS
{       (     SELECT      *
              FROM        "pr2" AS o2
              WHERE       o1.term = o2.term    )
    AND NOT EXISTS                 }
        (     SELECT      *
              FROM        "resPr" AS r
              WHERE       o1.term = r.term     );
```

Το μέρος της εντολής ανάμεσα στις αγκύλες αφαιρείται αν δεν υπάρχει στη βάση ο πίνακας για το *pr2* οπότε η πράξη ισοδυναμεί με απλή ενσωμάτωση της πληροφορίας του *pr1* στο *resPr*.

#### *4.4.2.10 Διαγραφή πίνακα*

Η εντολή που χρησιμοποιείται για την διαγραφή ενός πίνακα από τη βάση είναι:

*DROP TABLE IF EXISTS “name”;*

Εδώ δεν απαιτείται κάποια ενέργεια για την αφαίρεση του περιορισμού πρωτεύοντος κλειδιού καθώς αφαιρείται αυτόματα με την διαγραφή του πίνακα στον οποίο απευθύνεται.

#### *4.4.2.11 Διαγραφή γεγονότος*

Για τη διαγραφή ενός συγκεκριμένου γεγονότος από τη βάση π.χ. *pred('const')* ← θα δώσουμε την εντολή:

```
DELETE
FROM        “pred”
WHERE       term='const'
```

#### *4.4.2.12 Λήψη ονομάτων πινάκων*

Με αυτό το ερώτημα προς τη βάση λαμβάνουμε τα ονόματα όλων των πινάκων της.

```
SELECT      table_name
FROM        information_schema.tables
WHERE       table_schema='public'  AND  table_type='BASE TABLE';
```

Αυτή η εντολή διαφέρει από τις υπόλοιπες διότι αναφέρεται σε παραμέτρους της βάσης και όχι άμεσα σε στοιχεία που έχουν δημιουργήσει χρήστες. Το *information_schema* είναι ένα σύνολο όψεων (views) οι οποίες αναφέρονται στις παραμέτρους αυτές. Χρησιμοποιούμε την όψη *tables* όπου διατηρούνται πληροφορίες για όλους τους πίνακες και τις όψεις που αποτελούν την βάση. Θα χρειαστούμε τρεις στήλες. Την *table_type* με την οποία ξεχωρίζουμε τους μόνιμους πίνακες της βάσης, για τους οποίους ενδιαφερόμαστε, από τις υπόλοιπες εγγραφές οι οποίες περιλαμβάνουν όψεις, προσωρινούς πίνακες και ξένους πίνακες. Την *table_schema* η οποία δείχνει σε ποιο από τα σχήματα της βάσης περιέχεται κάθε πίνακας,

εδώ μας ενδιαφέρουν οι δημόσια διαθέσιμοι. Τέλος την στήλη *table_name* όπου θα βρούμε τα ονόματα όσων στοιχείων πληρούν τους προηγούμενους περιορισμούς.

### *4.4.3 Διαχειριστής βάσης SQL*

Η κλάση που υλοποιεί την διασύνδεση και τον χειρισμό της βάσης SQL είναι η *SQLDatabaseHandler*. Υλοποιεί την διεπαφή *DatabaseAccess* και προσφέρει ακριβώς τις ίδιες δημόσιες μεθόδους με την *RDFDadabaseHandler*.

Τα πεδία που ανασύρει ο κατασκευαστής της κλάσης από την παράμετρο τύπου *Configurator* είναι τα εξής:

- Το URL στο οποίο βρίσκεται η βάση, είναι της μορφής jdbc:postgresql://διεύθυνση server:θύρα/"όνομα βάσης"
- τα απαραίτητα στοιχεία για τη σύνδεση στη βάση: όνομα χρήστη και κωδικό πρόσβασης
- το αν η μεταβίβαση των εντολών ενημέρωσης της βάσης προς τον server θα γίνεται σε διαφορετικό νήμα
- το επίπεδο της λεπτομέρειας της περιγραφής των εσωτερικών διαδικασιών (verbosity)
- το αν θα καταγράφονται οι εντολές προς τη βάση και το αρχείο όπου θα γίνεται η καταγραφή

Αρχικά πραγματοποιεί τη σύνδεση με τη βάση, μέσω της οποίας θα μεταβιβαστούν οι εντολές. Το χαρακτηριστικό AutoCommit της σύνδεσης απενεργοποιείται. Αυτό έχει ως συνέπεια να μην οριστικοποιούνται οι αλλαγές στη βάση μετά την εκτέλεση κάθε εντολής. Οι εντολές στέλνονται σε ομάδες και οι αλλαγές οριστικοποιούνται μόνο μετά από την εκτέλεση κάθε τέτοιας ομάδας. Με αυτό τον τρόπο επιτυγχάνεται βελτίωση της ταχύτητας επεξεργασίας. Αν η ενημέρωση της βάσης έχει οριστεί να γίνεται παράλληλα σε αυτό το σημείο εκκινεί το νήμα με τον κατάλληλο κώδικα. Στη συνέχεια εκτελεί την πρώτη εντολή με την οποία ανακτά όλα τα ονόματα των πινάκων που περιέχονται στη βάση. Η απουσία πίνακα που να αντιστοιχεί σε κάποιο κατηγόρημα μας παρέχει έναν άμεσο τρόπο να γνωρίζουμε ότι δεν υπάρχει πληροφορία για το κατηγόρημα αυτό. Την πληροφορία αυτή εκμεταλλευόμαστε στην υλοποίηση της λειτουργίας πολλών από τις μεθόδους του επιτυγχάνοντας τον περιορισμό των εντολών που αποστέλλονται στον server. Αντιπροσωπευτικό παράδειγμα είναι η διαδικασία εκτέλεσης κανόνων. Προτού προχωρήσουμε στην δημιουργία της εντολής SQL που αντιστοιχεί στον προς εκτέλεση κανόνα και την μεταβίβασή της στη βάση ελέγχουμε αν υπάρχουν πίνακες για όλα τα κατηγορήματα που περιέχονται στο σώμα του. Αν

βρεθεί έστω και ένα κατηγόρημα για το οποίο δεν υπάρχει πληροφορία στη βάση, το σώμα του κανόνα καθίσταται αυτομάτως μη ικανοποιήσιμο. Αυτό έχει ως αποτέλεσμα να μην είναι δυνατή η παραγωγή νέας πληροφορίας από αυτόν τον και η εκτέλεσή του στη βάση είναι περιττή οπότε η διαδικασία τερματίζει εδώ και ο κανόνας αυτός απορρίπτεται. Με βάση το ίδιο σκεπτικό τροποποιούνται και οι περισσότεροι από τους υπόλοιπους τύπους εντολών. Είτε αποτρέπεται ολοκληρωτικά η εκτέλεσή τους είτε εκτελούνται μόνο ορισμένα τμήματά τους. Στη συνέχεια περιγράφεται ο τρόπος υλοποίησης των υπόλοιπων μεθόδων του interface *DatabaseAcess* για βάση τύπου SQL.

Όπως συμβαίνει και με τους κανόνες, η εκτέλεση του ερωτήματος δεν πραγματοποιείται αν δεν υπάρχουν πίνακες για όλα τα κατηγορήματα στο σώμα και επιστρέφεται απευθείας κενή απάντηση. Οι μέθοδοι *isAlreadyIn* και *delete* ελέγχουν πρώτα την ύπαρξη στη βάση του πίνακα που αντιστοιχεί στο κατηγόρημα του γεγονότος που έχουν σαν όρισμα και αν χρειαστεί τον δημιουργούν και εκτελούν τις εντολές SQL όπως αναφέρθηκαν παραπάνω. Ο έλεγχος για το αν περιέχεται πληροφορία σε ένα σύνολο κατηγορημάτων, που απαιτείται από την μέθοδο *areEmpty*, γίνεται μεμονωμένα για κάθε κατηγόρημα. Πρώτα ελέγχεται η ύπαρξη πίνακα και στην περίπτωση που βρεθεί υποβάλλεται το ερώτημα για να διαπιστωθεί η ύπαρξη ή όχι δεδομένων.

Για την υλοποίηση των υπόλοιπων μεθόδων της διεπαφής χρησιμοποιούνται οι παρακάτω βοηθητικές μέθοδοι: Οι *createTableIfNotYetCreated* και *dropTable* δημιουργούν και διαγράφουν αντίστοιχα τους πίνακες που αντιστοιχούν στα ορίσματά τους, αν φυσικά αυτό απαιτείται, ενημερώνοντας παράλληλα και τη λίστα με τους πίνακες που υπάρχουν στη βάση. Επειδή επιλέχθηκε να υπάρχει ξεχωριστός πίνακας για κάθε κατηγόρημα, προτού παραχθούν νέες πληροφορίες για κάποιο κατηγόρημα θα πρέπει να διασφαλιστεί ότι έχει δημιουργηθεί πίνακας στη βάση για αυτό. Για αυτόν το λόγο πριν από την εκτέλεση οποιασδήποτε εντολής που μπορεί να δημιουργήσει νέα δεδομένα πρέπει να εκτελεστεί η μέθοδος *createTableIfNotYetCreated* για να διασφαλιστεί ότι υπάρχει πίνακας όπου μπορούν να αποθηκευτούν. Αν αυτό δεν ισχύει, η εκτέλεση της εντολής είναι αδύνατη. Βέβαια, όπως είναι αναμενόμενο, η εκτέλεση μίας επιπλέον εντολής στη βάση πριν από κάθε κανόνα επιβαρύνει το σύστημα οπότε είναι σημαντικό να διασφαλιστεί ότι θα εκτελείται μόνο όταν είναι απολύτως απαραίτητο. Εδώ χρησιμεύει για ακόμα μία φορά η λίστα με τους υπάρχοντες πίνακες στη βάση την οποία συμβουλεύεται και αυτή η μέθοδος προτού επικοινωνήσει με τη βάση. Η βοηθητική μέθοδος *insertInto* εισάγει τα δεδομένα του κατηγορήματος πηγή στο κατηγόρημα προορισμό. Αν δεν υπάρχει η πηγή στη βάση δεν εκτελείται. Διαφορετικά δημιουργεί αν απαιτείται, μέσω της *createTableIfNotYetCreated*, τον πίνακα για το κατηγόρημα προορισμό και στέλνει την εντολή SQL για την αντιγραφή των δεδομένων της πηγής σε αυτόν.

Η union, που χρησιμοποιείται για την ένωση των πληροφοριών δύο κατηγορημάτων πηγών σε ένα τρίτο κατηγόρημα, καλεί την *insertInto* δύο φορές, μία για κάθε κατηγόρημα πηγή.

Για την υλοποίηση της *concatExistingInfoOnIdbPredicates*, η οποία χρησιμοποιείται για την αξιοποίηση της προϋπάρχουσας στη βάση πληροφορίας για IDB κατηγορήματα, καλείται και πάλι η *insertInto*, μία φορά για κάθε κατηγόρημα με τον προορισμό να αποτελείται από το όνομα του κατηγορήματος με το κατάλληλο πρόθεμα.

Η αφαίρεση των δεδομένων ενός κατηγορήματος που έχει το ρόλο του αφαιρετέου από ένα κατηγόρημα-μειωτέο με το αποτέλεσμα να αποθηκεύεται σε ένα τρίτο κατηγόρημα-προορισμό γίνεται ως εξής: Πρώτα ελέγχεται αν υπάρχει ο πίνακας του μειωτέου. Αν δεν υπάρχει η μέθοδος επιστρέφει χωρίς να πραγματοποιήσει αλλαγές. Στη συνέχεια ελέγχεται η ύπαρξη του αφαιρέτη. Αν δεν υπάρχει τότε καλείται η *insertInto* για να εισάγει όλες τις πληροφορίες που περιέχει ο μειωτέος στο κατηγόρημα-προορισμό. Διαφορετικά επιστρατεύεται η *createTableIfNotYetCreated* για να βεβαιωθούμε ότι υπάρχει ο πίνακας για το κατηγόρημα προορισμό και στη συνέχεια εκτελείται η εντολή SQL για την αφαίρεση που παρουσιάστηκε νωρίτερα.

Για την αλλαγή του ονόματος ενός κατηγορήματος αρχικά ελέγχεται η ύπαρξη πίνακα με το παλιό όνομα. Αν δεν βρεθεί τότε δεν πραγματοποιείται κάποια αλλαγή. Σε διαφορετική περίπτωση ελέγχεται αν υπάρχει πίνακας με το νέο όνομα και αν αυτός βρεθεί καλείται η *insertInto* για να μεταφερθούν τα δεδομένα του παλιού πίνακα ο οποίος κατόπιν διαγράφεται με κλίση της *dropTable*. Αν δεν υπάρχει ήδη πίνακας με το νέο όνομα τότε εκτελούνται οι εντολές μετονομασίας του πίνακα και του περιορισμού του πρωτεύοντος κλειδιού.

Οι μέθοδοι *deleteAllDataFrom*, *cleanupDB* και *nullifyDB* χρησιμοποιούν την βοηθητική μέθοδο *dropTable*. Η πρώτη με όρισμα το όνομα του δοσμένου κατηγορήματος, η *cleanupDB* με όσα ονόματα πινάκων περιέχουν το πρόθεμα που χαρακτηρίζει τις προσωρινές πληροφορίες (UPP) και η *nullifyDB* καλεί την *dropTable* για όλους τους πίνακες της βάσης.

Τέλος υπάρχει η μέθοδος *executeTranslatedCommand* η οποία εκτελεί απευθείας μια εντολή SQL στη βάση. Η μέθοδος αυτή παρέχεται για λόγους διευκόλυνσης του χρήστη που επιθυμεί να πειραματιστεί με τα διάφορα χαρακτηριστικά της βάσης πάνω στην οποία εργάζεται και δεν είναι απαραίτητη για την κανονική λειτουργία του προγράμματος. Προσοχή πρέπει να δοθεί στο γεγονός ότι εδώ δεν υφίσταται η προστασία της αυτόματης δημιουργίας πινάκων που εμφανίζονται στις εντολές. Έτσι είναι πολύ εύκολο ορθές εντολές να μην εκτελεστούν λόγω μη ύπαρξης κάποιου πίνακα.

### *4.4.4 Αποστολή εντολών στη βάση*

Οι εντολές SQL που εκτελούνται στη βάση χωρίζονται σε δύο κατηγορίες, τα ερωτήματα από τα οποία αναμένουμε ένα αποτέλεσμα και τις εντολές ενημέρωσης οι οποίες τροποποιούν τα περιεχόμενα της βάσης.

Η πλειοψηφία των μεθόδων βασίζονται σε εντολές ενημέρωσης. Η διαχείριση των εντολών αυτών γίνεται από την μέθοδο executeSqlUpdate η οποία δεν τις εκτελεί άμεσα αλλά τις ομαδοποιεί. Η εκτέλεσή τους θα πραγματοποιηθεί είτε όταν έρθει η ώρα να απαντηθεί κάποιο ερώτημα, οπότε και θα πρέπει να έχει ενημερωθεί η βάση, είτε όταν συγκεντρωθεί μεγάλος αριθμός εντολών και πλησιάζουμε στα όρια της διαθέσιμης μνήμης. Στην περίπτωση που η ενημέρωση της βάσης γίνεται σε ανεξάρτητο νήμα τότε η ομάδα των εντολών μεταβιβάζεται στο αντικείμενο που επιτελεί τη λειτουργία αυτή. Αν βρίσκεται σε εξέλιξη η επεξεργασία προηγούμενης ομάδας εντολών θα χρειαστεί να αναμείνουμε την ολοκλήρωσή της, όπως ακριβώς και στην περίπτωση της βάσης RDF.

Οι μέθοδοι που παράγουν ερωτήματα είναι οι *executeQuery*, *isEmpty*, *isAlreadyIn* και *getExistingTables*. Η *getExistingTables* χρησιμοποιεί προετοιμασμένο ερώτημα (prepared query) το οποίο προετοιμάζεται και μεταγλωττίζεται στον server την πρώτη φορά που καλείται η μέθοδος και επαναχρησιμοποιείται σε επόμενες κλήσεις της μεθόδου. Δυστυχώς αυτού του τύπου τα ερωτήματα δεν μπορούν να χρησιμοποιηθούν στις άλλες τρεις μεθόδους διότι απαιτείται να αναφέρονται σε διαφορετικούς πίνακες κάθε φορά και κάτι τέτοιο δεν υποστηρίζεται από τα προετοιμασμένα ερωτήματα. Τα ερωτήματα που παράγονται από αυτές τις μεθόδους δημιουργούνται σύμφωνα με τα ορίσματά τους κάθε φορά που καλούνται. Σε κάθε περίπτωση πριν την αποστολή ερωτήματος στη βάση πραγματοποιείται η εκτέλεση εντολών ενημέρωσης που μπορεί να αναμένουν να εκτελεστούν και αν αυτό γίνεται παράλληλα, σε διαφορετικό νήμα, και δεν έχει ολοκληρωθεί τότε το κύριο νήμα θα παγώσει μέχρι να πραγματοποιηθεί η ενημέρωση της βάσης.

## *4.5 Υποστηρικτικές κλάσεις*

Σε αυτή την ενότητα θα αναφερθούν μερικές από τις κλάσεις που υποστηρίζουν βασικές λειτουργίες του προγράμματος. Υπάρχει ένας αριθμός κλάσεων που δεν αναφέρεται καθώς οι λειτουργίες που επιτελούν είναι εκτός των πλαισίων του παρόντος κειμένου.

Predicate

Εδώ αποθηκεύονται οι πληροφορίες για κάθε κατηγόρημα που περιέχεται στους κανόνες της εισόδου. Περιλαμβάνει τα final πεδία *name, arity* και *adornment* και το Boolean flag *isIdbPredicate* το οποίο μπορεί να τροποποιηθεί. Στο πεδίο *arity* αποθηκεύεται ο αριθμός των όρων που δέχεται το κατηγόρημα, ένας αν πρόκειται για έννοια και δύο αν πρόκειται για ρόλο. Στο *adornment* αποθηκεύεται μία συμβολοσειρά από «b» και «f» που υποδεικνύουν δεσμευμένη ή ελεύθερη μεταβλητή στην αντίστοιχη θέση. Το *isIdbPredicate* τίθεται true αν κατά την διαδικασία δημιουργίας κάποιου κανόνα το κατηγόρημα βρεθεί στην κεφαλή. Οι μέθοδοι, πέρα από τις απαραίτητες seters/getters, περιλαμβάνουν τις *getPlainName()* και *getFullName()* με την πρώτη να επιστρέφει μόνο το όνομα του κατηγορήματος (πεδίο *name)* και τη δεύτερη το όνομα ακολουθούμενο από την κατάληξη «_» + «*adornment*», στην περίπτωση που το *adornment* υπάρχει. Τέλος έχουν παρακαμφθεί οι μέθοδοι *equals()* και *hashCode()*, με τις νέες να λαμβάνουν υπόψη μόνο τα final πεδία, για να μπορούν τα αντικείμενα τύπου *predicate* να τοποθετούνται σε δομές *HashSet* για την βελτιστοποίηση της απόδοσης του προγράμματος.

PredicatePool

Σκοπός της κλάσης αυτής είναι να διατηρεί μια συλλογή με όλα τα κατηγορήματα. Η συλλογή αυτή υλοποιείται από την *HashSet* για αποδοτικό έλεγχο για την ύπαρξη κάποιου κατηγορήματος σε αυτήν. Όταν απαιτηθεί η δημιουργία κάποιου κατηγορήματος ελέγχεται αυτή η συλλογή και αν υπάρχει ήδη κάποιο με τα χαρακτηριστικά που θέλουμε τότε επιστρέφεται αυτό. Έτσι όπου συναντάμε κατηγόρημα με συγκεκριμένο όνομα, πλήθος όρων και στολισμό πρόκειται για το αυτό αντικείμενο. Με αυτόν τον τρόπο πετυχαίνουμε σημαντικότατη εξοικονόμηση στον συνολικό αριθμό των αντικειμένων που διατηρούνται στη μνήμη. Επίσης διατηρώντας το ίδιο αντικείμενο μπορούμε να εντάξουμε σε αυτό πεδία όπως

το *isIdbPredicate* και να έχουμε άμεσο έλεγχο, αποφεύγοντας την διατήρηση στις κύριες κλάσεις κάποιας δομής όπου θα αποθηκεύονται τα IDB κατηγορήματα και θα αναζητείται αν περιέχεται σε αυτήν το κατηγόρημα που θέλουμε να ελέγξουμε. Απομακρύνεται δηλαδή από το προσκήνιο ο μηχανισμός για την υποστήριξη του ελέγχου για το αν κάποιο κατηγόρημα είναι IDB, με την ενημέρωση αυτού του πεδίου να γίνεται «αυτόματα» όταν κάποιο κατηγόρημα βρεθεί σε κεφαλή κανόνα όπως θα δούμε παρακάτω.

Η κλάση αυτή ακολουθεί το μοντέλο σχεδίασης singleton. Ο κατασκευαστής είναι private για να αποτραπεί η πρόσβαση σε αυτόν και η πρόσβαση στο μοναδικό αντικείμενο αυτού του τύπου γίνεται μέσω της *getPool()* η οποία θα δημιουργήσει την «πισίνα κατηγορημάτων» την πρώτη φορά που θα κληθεί. Στις περιπτώσεις που έχουμε μετασχηματισμό κανόνων, όπως για παράδειγμα στην *RulesAdornment*, τότε τα αρχικά κατηγορήματα παύουν να μας απασχολούν. Τότε καλείται η *drainPool()* για να αδειάσει την πισίνα πριν τον μετασχηματισμό. Έτσι πετυχαίνουμε ελάφρυνση της συλλογής και εξοικονόμηση χώρου αφού δεν θα υπάρχουν πλέον αναφορές στα παλιά κατηγορήματα τα οποία και θα διαγραφούν από τον μηχανισμό συλλογής απορριμμάτων της Java. Οι δύο κύριες μέθοδοί της είναι οι *getPredicate* και *getIdbPredicate* που καλούνται όταν χρειαζόμαστε κάποιο κατηγόρημα. Αν η «πισίνα» δεν περιέχει κάποιο με τα επιθυμητά χαρακτηριστικά τότε αυτό δημιουργείται και ταυτόχρονα αποθηκεύεται σε αυτήν. Η μόνη διαφορά μεταξύ τους είναι ότι η δεύτερη θέτει στο πεδίο *isIdbPredicate* την τιμή true. Τέλος υπάρχει και η μέθοδος *getIdbPredicatNames* που επιστρέφει τα ονόματα των IDB κατηγορημάτων. Χρησιμεύει στην περίπτωση που έχουμε χρησιμοποιήσει την τεχνική της μαγικής μεταγραφής για τον διαχωρισμό των προσωρινών κατηγορημάτων από τα αρχικά.

Atom

Αυτή είναι η κλάση των ατόμων, περιέχει το κατηγόρημα του ατόμου και έναν πίνακα με τους όρους του. Η μοναδική άξια λόγου, αν και αρκετά «τεχνική», μέθοδος της είναι η *getCopyOfAtomWithPrefixedName_bypassingPool* η οποία, όπως φανερώνει και το όνομά της, επιστρέφει ένα αντίγραφο του ατόμου το οποίο όμως έχει στο όνομα του κατηγορήματός του το πρόθεμα που δίνεται. Το αξιόλογο σημείο στην λειτουργία της είναι ότι παρακάμπτει την *PredicatePool*, δηλαδή δημιουργεί απευθείας νέο αντικείμενο τύπου *Predicate* αδιαφορώντας αν υπάρχει ήδη κάποιο με τις ίδιες ιδιότητες. Αυτό έχει ως συνέπεια να παραλείπεται ο έλεγχος για το αν περιέχεται το νέο κατηγόρημα στην συλλογή της *PredicatePool* πράγμα που επιταχύνει την διαδικασία. Το μειονέκτημα είναι ότι το πεδίο *isIdbPredicate* του νέου κατηγορήματος δεν μπορεί να ενημερωθεί στην συνέχεια για αυτό η

συγκεκριμένη μέθοδος χρησιμοποιείται μόνο κατά την παραγωγή κανόνων που μόνη χρήση έχουν την εκτέλεσή τους στη βάση, όπου δεν ενδιαφέρει η τιμή του πεδίου αυτού. Έτσι αποφεύγουμε την περιττή διαδικασία αναζήτησης στην *PredicatePool* για τέτοιου είδους κανόνες.

Fact

Τα γεγονότα είναι κανόνες datalog χωρίς σώμα. Η ανυπαρξία σώματος και των περιορισμών που αυτό επιβάλλει, έχει ως αποτέλεσμα η κεφαλή του να ισχύει πάντα. Επίσης δεν είναι δυνατό να περιέχει μεταβλητές αφού, λόγω της συνθήκης ασφαλείας, είναι απαραίτητο να εμφανίζονται και στο σώμα. Τα γεγονότα είναι η μικρότερη μονάδα πληροφορίας για ένα κατηγόρημα και αντιστοιχούν σε μία τριάδα RDF ή μια γραμμή σε πίνακα βάσης SQL. Μπορεί να συνοδεύουν τους κανόνες της εισόδου ως συμπληρωματική πληροφορία. Η πληροφορία αυτή εισάγεται στη βάση πριν ξεκινήσει η αποτίμηση του προγράμματος datalog.

DLRule

Πρόκειται για υποκλάση της Fact η οποία περιλαμβάνει πρόβλεψη και για σώμα. Τα αντικείμενα αυτής της κλάσης αναπαριστούν πλήρεις κανόνες datalog. Αρχικά σε αυτά αποθηκεύονται οι κανόνες της εισόδου. Για την υποστήριξη αυτής της λειτουργίας παρέχεται ένας κατασκευαστής, ο οποίος παίρνει ως παράμετρο την συμβολοσειρά που περιέχει τον κανόνα σε γραπτή μορφή και αναλαμβάνει να κάνει το parsing και να αποθηκεύσει στις κατάλληλες δομές τις πληροφορίες που αφορούν τον κανόνα αυτό. Η μέθοδος *makePrintable()* κάνει την αντίστροφη δουλειά και επιστρέφει μία συμβολοσειρά με τον κανόνα σε εκτυπώσιμη μορφή. Για την δημιουργία ενδιάμεσων κανόνων υπάρχει και η επιλογή του απλού κατασκευαστή που αρχικοποιεί ένα τέτοιο αντικείμενο το οποίο είναι έτοιμο να δεχθεί στη συνέχεια τις απαραίτητες πληροφορίες από τους διάφορους setters. Αυτό που αξίζει να σημειωθεί για αυτές τις μεθόδους είναι ότι αναλαμβάνουν να θέσουν true το πεδίο *isIdbPredicate* των κατηγορημάτων που βρίσκονται σε άτομα που καταλήγουν στην κεφαλή του κανόνα απαλλάσσοντας από αυτή την ευθύνη τις κλάσεις που χρησιμοποιούν τους *DLRules* οι οποίες βρίσκουν ενημερωμένο το συγκεκριμένο πεδίο.

ReadRules

Αυτή η κλάση αναλαμβάνει να εισάγει στην εφαρμογή τους κανόνες από το αρχείο της μεταγραφής. Πέρα από το Rapid υποστηρίζονται μεταγραφές και από το Clipper. Παρακάμπτει τα σχόλια και αναγνωρίζει τις γραμμές που περιέχουν κανόνες. Επιπλέον ελέγχει για ατέλειες στις μεταγραφές όπως για κανόνες που παρουσιάζονται περισσότερες από μία φορές. Τέλος εντοπίζει το ερώτημα, αν υπάρχει και διαχωρίζει τα γεγονότα από τους υπόλοιπους κανόνες.

UniqueStringCreator

Η κλάση *UniqueStringCreator* έχει διττό ρόλο. Αναλαμβάνει να δημιουργήσει μία τυχαία συμβολοσειρά που θα χρησιμοποιηθεί ως UPP (Unique Processing Prefix) ή αν έχει ήδη οριστεί μία τέτοια θα την ελέγξει για να πιστοποιηθεί ότι πράγματι πληροί την απαίτηση της μοναδικότητας. Αν αυτό δεν συμβαίνει θα επεκτείνει τη δοσμένη συμβολοσειρά με τέτοιο τρόπο ώστε να μπορεί θα χρησιμοποιηθεί ως UPP στη συνέχεια. Όπως έχει ήδη αναφερθεί το UPP χρησιμοποιείται για να καταστεί δυνατός ο διαχωρισμός στη βάση δεδομένων των προσωρινών πληροφοριών και των ενδιάμεσων αποτελεσμάτων από τα τελικά και τις προϋπάρχουσες πληροφορίες. Για αυτό απαιτείται να μην περιλαμβάνεται σε κάποιο όνομα κατηγορήματος. Άρα πρέπει να συγκριθεί με δύο ομάδες ονομάτων. Η πρώτη αποτελείται από όλα τα πεδία της βάσης όπου μπορεί να περιέχονται ονόματα κατηγορημάτων και η δεύτερη από τα ονόματα των IDB κατηγορημάτων που περιέχονται στους κανόνες της εισόδου. Τα EDB κατηγορήματα των κανόνων της εισόδου δεν έχει νόημα να συμπεριληφθούν στον έλεγχο αφού, λόγω της θέσης τους στο σώμα των κανόνων, δεν πρόκειται να παραχθεί πληροφορία για αυτά. Είναι όμως απαραίτητο να ελεγχθούν τα ονόματα των EDB κατηγορημάτων για τα οποία προϋπάρχει πληροφορία στη βάση. Αυτά θα έρθουν για έλεγχο με την πρώτη ομάδα ονομάτων. Τα γεγονότα που μπορεί να περιλαμβάνονται στα δεδομένα της εισόδου πρέπει να έχουν ήδη ενσωματωθεί στη βάση προτού κληθεί η *UniqueStringCreator* για να συμπεριληφθούν και τα κατηγορήματα στα οποία αναφέρονται στον έλεγχο.

Το χτίσιμο του UPP γίνεται σταδιακά, όσο προχωράει η διαδικασία του ελέγχου. Ξεκινάμε με την επιλογή του πρώτου χαρακτήρα με τυχαίο τρόπο από ένα σύνολο διαθέσιμων χαρακτήρων και προχωράμε στην σύγκριση με τα ονόματα των κατηγορημάτων. Αν βρεθεί

κάποιο που να τον περιέχει επιλέγεται και δεύτερος χαρακτήρας και ο έλεγχος επαναλαμβάνεται. Αν περιέχεται και η νέα συμβολοσειρά των δύο χαρακτήρων τότε γίνεται νέα «κλήρωση» και ο χαρακτήρας που επιλέγεται αντικαθιστά τον δεύτερο. Η διαδικασία επιλογής νέου χαρακτήρα θα επαναληφθεί όσες φορές χρειαστεί μέχρι να βρεθεί συνδυασμός που δεν περιέχεται στο όνομα που την ξεκίνησε. Υπάρχει μια μικρή πιθανότητα να εισέλθουμε σε ατέρμονα βρόχο αν συναντήσουμε κάποιο πολύ ιδιαίτερο όνομα που θα εξαντλεί τις διαθέσιμες επιλογές χαρακτήρων. Αν για παράδειγμα αρχικά είχαμε τη συμβολοσειρά «a», διαθέσιμα σύμβολα τα [a, b, c] και το όνομα «aabac» το οποίο περιέχει όλους τους διαθέσιμους συνδυασμούς το πρόγραμμα δεν θα μπορούσε να προχωρήσει. Για αυτό υπάρχει ένας μετρητής των αποτυχημένων προσπαθειών ο οποίος σε συγκεκριμένα διαστήματα προκαλεί την αύξηση κατά ένα του αριθμού των νέων χαρακτήρων που επιλέγονται. Στο παράδειγμά μας θα μπορούσε να επιλεγεί το «cc» για να προστεθεί στο «a». Το πρώτο μέρος της συμβολοσειράς δεν αλλάζει αφού εξασφαλίζει την μοναδικότητα σε σχέση με τα ονόματα που έχουν ήδη ελεγχθεί. Τα παραπάνω ισχύουν και για την περίπτωση που έχει δοθεί μία αρχική συμβολοσειρά, με τη διαφορά ότι ξεκινάμε με αυτήν αντί του πρώτου τυχαίου χαρακτήρα.

Για να διασφαλιστεί η μέγιστη δυνατή συμβατότητα σε μελλοντικές επεκτάσεις του συνόλου των υποστηριζόμενων συστημάτων διαχείρισης βάσεων δεδομένων, ο πρώτος χαρακτήρας επιλέγεται από το σύνολο των μικρών λατινικών γραμμάτων και οι υπόλοιποι από το σύνολο αυτό προσαυξημένο με τους αριθμούς και την κάτω παύλα. Όπως αποδείχθηκε και στην πράξη οι χαρακτήρες αυτοί επαρκούν για τη δημιουργία αρκετά μικρών μοναδικών συμβολοσειρών. Υπάρχει η δυνατότητα τροποποίησης του συνόλου των διαθέσιμων χαρακτήρων μέσω της μεθόδου *setAvailableSymbols*.

AnswerFormatter

Η κλάση αυτή μπορεί προαιρετικά να χρησιμοποιηθεί για να διαμορφωθούν οι πληροφορίες που επιστρέφονται από την εκτέλεση του ερωτήματος στη βάση σε μορφή κατάλληλη για εκτύπωση σε αρχείο ή παρουσίαση στην οθόνη. Διαχωρίζει τις στήλες με τις τιμές των μεταβλητών οι οποίες παρουσιάζονται ομοιόμορφα στοιχισμένες σε μορφή πίνακα.

Configurator

Εδώ συγκεντρώνονται διάφορες παράμετροι που είναι απαραίτητες για την εκτέλεση. Ενδεικτικά εδώ αποθηκεύονται:

- Το αρχείο όπου θα αποθηκευτεί η απάντηση στην περίπτωση που υπάρχει κάποιο ερώτημα. Αν δεν δοθεί κάποιο αρχείο τότε η απάντηση εμφανίζεται στην οθόνη.
- Το πρόθεμα που τυχόν έχουν τα κατηγορήματα στη βάση.
- Ο τρόπος με τον οποίο θέλουμε να ενημερωθεί η βάση μετά το πέρας της εκτέλεσης.
- Το υποψήφιο UPP, που θα εξελιχθεί σε μία συμβολοσειρά που δεν υπάρχει σε κάποιο όνομα κατηγορήματος για να ξεχωρίζουν οι νέες πληροφορίες που αποθήκευσε το πρόγραμμα στη βάση.
- Τα απαραίτητα στοιχεία για τη σύνδεση στη βάση όπως ο τύπος της, η διεύθυνση URL στην οποία βρίσκεται και τα στοιχεία ταυτοποίησης χρήστη αν απαιτούνται.
- Προαιρετικά μπορεί να δοθεί ένα αρχείο στο οποίο θα αποθηκευθούν στοιχεία σχετικά με την εκτέλεση όπως η χρονική διάρκεια κάθε επιμέρους βήματος και διάφορες παράμετροι όπως ο αριθμός των νημάτων που χρησιμοποιήθηκαν .
- Ένα Boolean πεδίο με αρχική τιμή false που τίθεται true από την Magic Rewriting αν αυτή εκτελεστεί. Απαιτείται για δύο λόγους: Πρώτον γιατί η τεχνική αυτή προσθέτει νέα βοηθητικά κατηγορήματα τα οποία δεν πρέπει να μείνουν στη βάση αν έχει επιλεγεί να ενσωματωθεί η παραχθείσα γνώση. Οπότε ελέγχοντας την τιμή του μπορούμε να αποφασίσουμε ποιά διαδικασία θα ακολουθηθεί για την ενσωμάτωση της παραχθείσας πληροφορίας στη βάση. Δεύτερον διότι διαφοροποιείται ριζικά ο τρόπος με τον οποίο αξιοποιούνται οι πληροφορίες που μπορεί να προϋπάρχουν στη βάση για κατηγορήματα IDB με δεσμευμένες μεταβλητές στο adornment τους.
- Ένα πεδίο που καθορίζει πόσο λεπτομερείς είναι οι πληροφορίες που δίνονται στην πρότυπη έξοδο κατά τη διάρκεια της εκτέλεσης του προγράμματος (*verbosity*). Παρέχονται τέσσερα διαφορετικά επίπεδα.
- Μία σημαία για την ενεργοποίηση της λειτουργίας καταγραφής των εντολών που στέλνονται στη βάση και το αρχείο στο οποίο θα αποθηκευτεί η λίστα των εντολών.

Εδώ γίνεται και η κατανομή των διαθέσιμων υπολογιστικών πόρων στις διάφορες λειτουργίες. Συγκεκριμένα καθορίζονται:

- Αν θα γίνεται σε ξεχωριστό νήμα η εκτέλεση των εντολών ενημέρωσης της βάσης. Αυτή είναι η πρώτη λειτουργία που εκτελείται σε νέο νήμα, όταν οι διαθέσιμοι υπολογιστικοί πόροι το επιτρέψουν, δεδομένου ότι η ενημέρωση της βάσης είναι μια χρονοβόρα διαδικασία, ειδικά αν τα δεδομένα διατηρούνται εκτός της κύριας μνήμης.

- Ο αριθμός των νημάτων για την παράλληλη δημιουργία των προσωρινών κανόνων.
- Αν ο αριθμός των διαθέσιμων επεξεργαστών το επιτρέπει ενεργοποιείται και η παράλληλη μετάφραση κανόνων σε εντολές ενημέρωσης προς την βάση.
- Καθορίζει τον τύπο των ενδιάμεσων κανόνων ανάλογα με το είδος της χρησιμοποιούμενης βάσης.

## *4.6 Έλεγχος ορθότητας*

Για την κάλυψη της ανάγκης για επαλήθευση της ορθότητας του προγράμματος δημιουργήθηκε μια εφαρμογή που χρησιμοποιεί το πρόγραμμα ως API. Χρησιμοποιήθηκε επανειλημμένα στα τελευταία στάδια της ανάπτυξης καθώς ελέγχει το τελικό αποτέλεσμα που δίνεται από το πρόγραμμα, το οποίο απαιτεί την λειτουργία όλων των υποσυστημάτων. Συνέβαλλε καθοριστικά, πέρα από την ανάπτυξη και στον έλεγχο ιδεών για την βελτίωση της απόδοσης. Επιπλέον πιστοποίησε την ορθότητα των μεταγραφών του Rapid σε όλες τις δοκιμές.

Αναλαμβάνει να τρέξει μια σειρά από παραδείγματα για τα οποία υπάρχει γνωστή απάντηση επαληθευμένη με άλλα μέσα. Τα παραδείγματα αυτά είναι οργανωμένα με συγκεκριμένο τρόπο και μπορεί να γίνει επιλεκτική εκτέλεση όσων ενδιαφέρουν κάθε φορά. Η εφαρμογή αυτή αναλαμβάνει τον υπολογισμό της τελικής απάντησης και την επαλήθευσή της ορθότητάς της μέσω της σύγκρισής της με την απάντηση αναφοράς.

Τα παραδείγματα είναι οργανωμένα σε υποφακέλους μέσα σε έναν κεντρικό φάκελο. Παραδείγματα αναζητούνται σε όλους τους υποφακέλους εκτός από αυτούς των οποίων το όνομα ξεκινάει με τον χαρακτήρα «_». Για παράδειγμα υπάρχει ο φάκελος «_Stats» όπου αποθηκεύονται στοιχεία σχετικά με την εκτέλεση του προγράμματος όπως ο χρόνος που απαιτήθηκε από τα διάφορα υποσυστήματα και οι ρυθμίσεις με τις οποίες έγινε η εκτέλεση. Τέσσερα είναι τα σημεία ενδιαφέροντος στους φακέλους που περιέχουν παραδείγματα:

- Ο φάκελος «Queries» όπου βρίσκονται τα ερωτήματα που αφορούν το συγκεκριμένο παράδειγμα. Κάθε ερώτημα περιλαμβάνεται σε ένα αρχείο κειμένου.
- Ο φάκελος «Rewritings» όπου βρίσκονται, επίσης σε ξεχωριστά αρχεία, οι διαθέσιμες μεταγραφές από τα υποστηριζόμενα προγράμματα.
- Ο φάκελος «Answers» στον οποίο θα αποθηκευθούν οι απαντήσεις του προγράμματος αλλά βρίσκονται ήδη και οι πρότυπες απαντήσεις με τις οποίες θα γίνει η επαλήθευση.

- Το αρχείο «config.txt» το οποίο περιλαμβάνει τις απαραίτητες παραμέτρους για την διαδικασία της επαλήθευσης.

Η περιγραφή της λειτουργίας του αρχείου «config.txt» θα γίνει ευκολότερα μέσω ενός παραδείγματος:

**Εικόνα 3: Παράδειγμα αρχείου «config.txt».**

```
queries::Query_?.txt
Το ερωτηματικό μπαίνει στη θέση του αριθμού/αναγνωριστικού
rapid_rewritings::rapid_Q?.txt
clipper_rewritings::datalogOnLUBM_Q?.txt
answers_rapidRW::myAns_?.txt
answers_clipperRW::myAns_?onClipperRW.txt
template_answers::datalogOnLUBM_Q?.txt-answer.txt
facts_of_ABox:: factsFrom_datalogOnLUBM_Q?.txt
skip_queries::a00, 10,11,12,12b
```

Όπως βλέπουμε περιέχει διάφορα πεδία με τα ονόματα και τις τιμές τους διαχωρισμένα με «::» και ελεύθερο κείμενο. Τα πρώτα επτά πεδία, με τη σειρά που εμφανίζονται, καθορίζουν τη μορφή που έχουν τα ονόματα των αρχείων που περιέχουν τα ερωτήματα, τις μεταγραφές από το Rapid και το Clipper, τις απαντήσεις που θα δώσει το πρόγραμμα με βάση αυτές τις μεταγραφές, οι πρότυπες απαντήσεις και τα αρχεία όπου μπορούν να περιέχονται τα γεγονότα του ABox. Το ερωτηματικό «?» είναι ο ειδικός χαρακτήρας που μπαίνει στο σημείο όπου βρίσκεται το αναγνωριστικό το οποίο δείχνει σε ποια υποπερίπτωση του παραδείγματος αναφέρεται το κάθε αρχείο. Για παράδειγμα σύμφωνα με το παραπάνω αρχείο το ερώτημα 3B βρίσκεται στο αρχείο «Query_3B.txt», η αντίστοιχη μεταγραφή από το Rapid στο αρχείο «rapid_Q3B.txt» και η απάντηση θα αποθηκευτεί στο αρχείο «myAns_3B.txt». Τέλος το πεδίο «skip_queries» υποδεικνύει για ποια ερωτήματα δεν ενδιαφέρει να υπολογιστεί απάντηση.

Η διαδικασία του ελέγχου ξεκινά με την διαδοχική διερεύνηση των φακέλων των παραδειγμάτων. Για κάθε παράδειγμα λαμβάνονται οι πληροφορίες από το αρχείο «config.txt» που το συνοδεύει και στη συνέχεια ελέγχονται ένα προς ένα τα ερωτήματα για αυτό. Αν δεν έχουμε να κάνουμε με ερώτημα που βρίσκεται στη λίστα προς παράλειψη, αναζητούνται τα διαθέσιμα rewritings. Πριν προχωρήσουμε στον υπολογισμό και την επαλήθευση της απάντησης γίνεται διαγραφή όλων των περιεχομένων της βάσης για να μην επηρεάζεται η απάντηση από δεδομένα που ανήκουν σε προηγούμενο ερώτημα. Τα γεγονότα

του ABox είτε συμπεριλαμβάνονται στο rewriting, είτε φορτώνονται αυτόματα από ξεχωριστό αρχείο, είτε δίνεται η δυνατότητα να φορτωθούν στη βάση από τον χρήστη πριν ξεκινήσει η αποτίμηση της απάντησης. Επίσης διαγράφονται υπάρχοντα αρχεία απάντησης από προηγούμενη εκτέλεση της εφαρμογής ελέγχου ορθότητας για να μην επηρεαστούν τα αποτελέσματα την τρέχουσας δοκιμής.

Σε αυτό το σημείο, ανάλογα με το αν επιθυμούμε να γίνει κορεσμός της βάσης ή απλώς να δοθεί απάντηση στο ερώτημα, ο έλεγχος περνά σε μία εκ των SaturateAndAnswerToFile ή CalculateAnswerToFile. Η πρώτη θα αναλάβει να δημιουργήσει το κατάλληλο αντικείμενο της κλάσης Configurator και θα προσπαθήσει να θέσει ως αληθές το πεδίο «tstSatAskOriginalQuery» αυτού. Αυτό έχει ως αποτέλεσμα μετά το πέρας του υπολογισμού να σταλεί στη βάση το ερώτημα χωρίς να τροποποιηθούν τα ονόματα των κατηγορημάτων του. Έτσι μπορούμε να ελέγξουμε και την διαδικασία ενσωμάτωσης των τελικών αποτελεσμάτων στη βάση. Η CalculateAnswerToFile θα υπολογίσει την απάντηση στο ερώτημα αξιοποιώντας τη δυνατότητα φιλτραρίσματος και απόρριψης των κανόνων που δεν συντελούν στο να προστεθεί νέα πληροφορία σε κάποιο από τα κατηγορήματα που μετέχουν σε αυτό. Αν έχει επιλεγεί να ακολουθηθεί η διαδρομή της μαγικής μεταγραφής για τον υπολογισμό του αποτελέσματος σε αυτό το σημείο θα κληθεί η RulesAdornment η οποία θα «στολίσει» τους κανόνες που ενδιαφέρουν για την απάντηση στο ερώτημα και θα απορρίψει τους εναπομείναντες κανόνες. Μετά αναλαμβάνει η MagicRewriting να δημιουργήσει τη μεταγραφή. Στη συνέχεια, είτε είχαμε την μαγική μεταγραφή των κανόνων είτε όχι, θα κληθεί η MutualRecursionEquivalence για να διαχωρίσει τους κανόνες σε ομάδες με στόχο την αποδοτικότερη αποτίμησή τους και τέλος η DatalogEvaluator για να υπολογίσει την απάντηση. Κατά τη διάρκεια του υπολογισμού η CalculateAnswerToFile συλλέγει πληροφορίες για την πορεία του υπολογισμού και τον χρόνο που διαρκεί κάθε βήμα. Αν έχει ζητηθεί να γίνει επαλήθευση της απάντησης και υπάρχει πρότυπη απάντηση για αυτό το ερώτημα τότε εκτελείται η AnswerVerifier για να συγκρίνει με αυτή την απάντηση που μόλις δόθηκε από το προηγούμενο βήμα. Αυτή η κλάση έχει τις κατάλληλες μεθόδους για να μπορεί να διαβάσει τα διαφορετικά αρχεία των απαντήσεων και να τα συγκρίνει γραμμή προς γραμμή. Ενημερώνει για πληροφορίες που μπορεί να μην υπάρχουν και στα δύο αρχεία, πράγμα ιδιαίτερα βοηθητικό στην αποσφαλμάτωση. Η παραπάνω διαδικασία επαναλαμβάνεται για όλα τα ερωτήματα και παραδείγματα.

# 5

# *Αποτελέσματα*

Λόγω των στενών δεσμών μεταξύ της εφαρμογής και του Rapid η παρουσίαση των αποτελεσμάτων δεν θα μπορούσε να είναι ολοκληρωμένη αν δεν αναφερθεί σε όλο το εύρος των τεχνολογιών που εμπλέκονται. Για το λόγο αυτό θα παρουσιαστεί διαδικασία αποτίμησης ερωτημάτων σε βάσεις γνώσης μέσω του Rapid και της υλοποιηθήσας εφαρμογής συνολικά. Με αυτό το δεδομένο, το σημείο εκκίνησης είναι οι οντολογίες. Θα παρουσιαστούν αποτελέσματα για δύο οντολογίες, την LUBM και την GALEN. Η πρώτη οντολογία αναφέρεται στο πεδίο της πανεπιστημιακής εκπαίδευσης και προέρχεται από το εργαλείο συγκριτικής ανάλυσης συστημάτων βάσεων γνώσης του Πανεπιστημίου Lehigh. Το συγκεκριμένο εργαλείο είναι ευρέως διαδεδομένο και αποτελεί ίσως το σημαντικότερο μέτρο σύγκρισης μεταξύ διαφορετικών συστημάτων. Η GALEN είναι μια ιδιαίτερα εκτεταμένη οντολογία που αναφέρεται στην ιατρική ορολογία. Πρόκειται για μία από τις δυσκολότερες δοκιμασίες στις οποίες μπορούν να υποβληθούν τα συστήματα συλλογιστικής καθώς περιλαμβάνει 23141 κλάσεις και 950 ιδιότητες. Και οι δύο οντολογίες είναι γραμμένες σε OWL.

Θα χρησιμοποιήσουμε επτά ερωτήματα με την οντολογία LUBM και δύο με την GALEN. Τα ερωτήματα για τις δύο οντολογίες καταγράφονται στους παρακάτω πίνακες, με τη μορφή που χρησιμοποιήθηκαν:

**Πίνακας 6: Ερωτήματα προς τη GALEN**

| | |
|---|---|
| GALEN a | Q(?0) <- PlateletCountProcedure(?0) |
| GALEN b | Q(?0) <- HaemoglobinConcentrationProcedure(?0) |

**Πίνακας 7: Ερωτήματα προς τη LUBM**

| | |
|---|---|
| LUBM a | Q(?0,?44) <- GraduateStudent(?0), takesCourse(?0, ?44) |
| LUBM b | Q(?0,?44) <- Publication(?0), publicationAuthor(?0,?44) |
| LUBM c | Q(?0,?1,?2,?3,?44) <- Professor(?0), worksFor(?0, ?44), name(?0,?1), emailAddress(?0,?2), telephone(?0, ?3) |
| LUBM d | Q(?0,?44) <- Person(?0), memberOf(?0, ?44) |
| LUBM e | Q(?0) <- Student(?0) |
| LUBM f | Q(?0,?44) <- ResearchGroup(?0), subOrganizationOf(?0, ?44) |
| LUBM g | Q(?0,?44) <- Person(?0), hasAlumnus(?44, ?0) |

Αρχικά εκτελείται το Rapid, μία φορά για κάθε ερώτημα. Η είσοδός του συμπληρώνεται με το αρχείο της αντίστοιχης οντολογίας. Οι μεταγραφές που δίνει ποικίλουν σε μέγεθος. Υπάρχουν απλά ερωτήματα, όπως αυτά που ανασύρουν τις πληροφορίες μιας συγκεκριμένης απλής έννοιας ή ρόλου. Οι μεταγραφές αυτών των ερωτημάτων αποτελούνται από το ίδιο το ερώτημα, μπορούμε να πούμε δηλαδή ότι εκφυλίζονται σε ένα συζευκτικό ερώτημα. Ένα παράδειγμα μεταγραφής αυτού του είδους συναντάμε στο πρώτο ερώτημα για τη LUBM. Ένα παράδειγμα μεταγραφής σε πλήρες πρόγραμμα datalog, με περιορισμένη έκταση που επιτρέπει την παρουσίασή του εδώ, είναι η μεταγραφή του δεύτερου ερωτήματος για τη LUBM:

**Πίνακας 8: Η έξοδος του Rapid για το δεύτερο ερώτημα προς τη LUBM.**

```
@q(?0, ?44) <- publication(?0), publicationauthor(?0, ?44)
article(?v1) <- technicalreport(?v1)
article(?v1) <- journalarticle(?v1)
article(?v1) <- conferencepaper(?v1)
publication(?v1) <- orgpublication(?_u1, ?v1)
publication(?v1) <- publicationauthor(?v1, ?_u3)
publication(?v1) <- publicationdate(?v1, ?_u5)
publication(?v1) <- publicationresearch(?v1, ?_u4)
```

```
publication(?v1) <- softwaredocumentation(?_u2, ?v1)
publication(?v1) <- unofficialpublication(?v1)
publication(?v1) <- article(?v1)
publication(?v1) <- specification(?v1)
publication(?v1) <- book(?v1)
publication(?v1) <- manual(?v1)
publication(?v1) <- software(?v1)
q(?0, ?44) <- publication(?0), publicationauthor(?0, ?44)
software(?v1) <- softwaredocumentation(?v1, ?_u6)
software(?v1) <- softwareversion(?v1, ?_u7)
```

Το μέγεθος της μεταγραφής είναι ένας σημαντικός δείκτης για την πολυπλοκότητα της διαδικασίας αποτίμησης. Όσο αυξάνεται το πλήθος των κανόνων και των IDB κατηγορημάτων η αποτίμηση καθίσταται δυσκολότερη και πιο χρονοβόρα.

Βέβαια αυτό δεν ισχύει πάντα. Κατά την διάρκεια των δοκιμών του προγράμματος διαπιστώθηκε ότι κάποια τρίτα εργαλεία που πραγματοποιούν μεταγραφή ερωτημάτων σε προγράμματα datalog δεν επιστρέφουν βελτιστοποιημένες μεταγραφές. Σε αυτά κωδικοποιείται κάθε φορά το σύνολο της οντολογίας στο πρόγραμμα datalog. Αυτό που διαφοροποιεί τις μεταγραφές μεταξύ τους είναι η προσθήκη ενός επιπλέον κανόνα που αντιστοιχεί στο μεταγραφόμενο ερώτημα. Αντιθέτως, το Rapid πραγματοποιεί στοχευμένη μεταγραφή. Το αποτέλεσμα είναι οι μεταγραφές διαφορετικών ερωτημάτων προς την ίδια οντολογία να παρουσιάζουν σημαντικές διαφοροποιήσεις. Μεταγραφές που διαφέρουν μόνο σε έναν κανόνα, αυτόν του ερωτήματος, επιστρέφονται αν τα ερωτήματα απευθύνονται σε κατηγορήματα που ανήκουν στο ίδιο τμήμα του συνολικού προγράμματος datalog που αντιστοιχεί σε ολόκληρη την οντολογία. Τα τμήματα για τα οποία γίνεται λόγος είναι αυτά που επιστρέφονται από τη διαδικασία της τμηματοποίησης που παρουσιάστηκε στο κεφάλαιο της datalog. Παράδειγμα τέτοιου είδους ερωτημάτων είναι τα ερωτήματα a και b προς τη GALEN. Τα κατηγορήματα ή εναλλακτικά κλάσεις στην ορολογία της OWL, HaemoglobinConcentrationProcedure και LymphocyteCountProcedure βρίσκονται στο ίδιο τμήμα του προγράμματος datalog που αντιστοιχεί στο σύνολο της οντολογίας GALEN. Για να γίνει αντιληπτή η έκταση της βελτίωσης που επιτυγχάνεται από το Rapid αρκεί να αναφερθεί ότι η έκταση της μεταγραφής που παράγεται από ένα μη βελτιστοποιημένο σε αυτόν τον τομέα εργαλείο για τα ερωτήματα προς τη GALEN είναι 51405 κανόνες! Ένα τόσο μεγαλύτερο νούμερο επιβαρύνει αισθητά, πέρα από το ίδιο το πρόγραμμα που δημιουργεί τη μεταγραφή και τη διαδικασία αποτίμησης. Παρά την ύπαρξη των μηχανισμών απόρριψης περιττών κανόνων, η διαφορά στο χρόνο εκτέλεσης της αποτίμησης μιας τέτοιας μεταγραφής είναι σχετικά μεγάλη.

Σε ότι αφορά την διαδικασία αποτίμησης, τη σκυτάλη σε αυτό το σημείο παίρνει η εφαρμογή αποτίμησης της μεταγραφής. Πέρα από τη μεταγραφή, τώρα λαμβάνεται υπόψη και το σύνολο δεδομένων που συνοδεύει την οντολογία. Στον παρακάτω πίνακα παρουσιάζονται οι χρόνοι αποτίμησης των μεταγραφών που αντιστοιχούν στα διάφορα ερωτήματα.

**Πίνακας 9: Χαρακτηριστικά μεγέθη αποτίμησης μεταγραφών.**

| Ερώτημα | Χρόνος αποτίμησης σε RDF (ms) | Χρόνος αποτίμησης σε SQL (ms) | Μέγεθος μεταγραφής (κανόνες) | Πλήθος εντολών ενημέρωσης από κανόνες |
|---|---|---|---|---|
| LUBM a | 131 | 187 | 1 | 2 |
| LUBM b | 120 | 109 | 17 | 4 |
| LUBM c | 249 | 1187 | 96 | 188 |
| LUBM d | 202 | 891 | 96 | 40 |
| LUBM e | 353 | 2096 | 31 | 394 |
| LUBM f | 50 | 63 | 2 | 4 |
| LUBM g | 142 | 687 | 96 | 6 |
| GALEN a | 385 | 2640 | 116 | 438 |
| GALEN b | 424 | 2859 | 116 | 508 |

Στις δύο τελευταίες στήλες του πίνακα βλέπουμε το μέγεθος της μεταγραφής σε κανόνες και τον αριθμό των κανόνων που έφτασαν μέχρι το σημείο της εκτέλεσης στη βάση. Σε ορισμένες περιπτώσεις παρατηρούμε ότι ο αριθμός των εκτελεσθέντων κανόνων είναι μικρότερος από τον αριθμό των κανόνων στη μεταγραφή. Αυτό οφείλεται στον έλεγχο που εφαρμόζεται για να εξακριβωθεί αν υπάρχουν δεδομένα στη βάση για κάθε κατηγόρημα. Συνέπεια αυτής της τεχνικής βελτιστοποίησης της αποτίμησης είναι η μη εκτέλεση κανόνων που περιλαμβάνουν στο σώμα κενά κατηγορήματα. Η εκτέλεσή τους δε θα άλλαζε τα περιεχόμενα της βάσης αφού δεν είναι δυνατό να ικανοποιηθούν οι περιορισμοί που κωδικοποιούνται στο σώμα τους. Τα σύνολα δεδομένων που χρησιμοποιήθηκαν για την αποτίμηση δεν περιλαμβάνουν πληροφορίες για όλα τα κατηγορήματα της οντολογίας. Πρέπει να αναφερθεί ότι τα σύνολα δεδομένων ήταν πανομοιότυπα και για τις δύο βάσεις.

Η αποτίμηση και για τις δύο βάσεις πραγματοποιήθηκε στο ίδιο μηχάνημα, με τετραπύρηνο επεξεργαστή Intel Xeon E5450. Οι ρυθμίσεις της εφαρμογής ήταν οι βέλτιστες για κάθε βάση. Για τον περιορισμό της επίδρασης της επικοινωνίας με τη βάση στους τελικούς χρόνους και οι δύο βάσεις δεδομένων έτρεχαν στον ίδιο υπολογιστή. Ο εξυπηρετητής Fuseki είχε ρυθμιστεί να διατηρεί το σύνολο δεδομένων στην κύρια μνήμη ενώ ο PostreSQL ήταν εγκαταστημένος σε εικονικό τοπικό δίσκο που βρισκόταν στη μνήμη (ramdisk).

Σε ότι αφορά τους χρόνους, σαφώς καλύτερες επιδόσεις σημείωσε ο Fuseki. Σε μεταγραφές όπου δεν υπάρχουν ή εκτελούνται πολλοί κανόνες η διαφορά του με τον PostgreSQL είναι μικρή. Όμως όσο μεγαλώνουν οι απαιτήσεις για εκτέλεση κανόνων στη βάση τόσο διευρύνεται η διαφορά μεταξύ των δύο τεχνολογιών. Μπορεί να βιαστεί κανείς και να μιλήσει για μεγαλύτερη καταλληλότητα της τεχνολογίας των αποθηκών τριάδων RDF για την υποστήριξη συστημάτων συλλογιστικής. Βέβαια, αυτό ισχύει ως ένα βαθμό αφού ο περιορισμός των δύο όρων στα κατηγορήματα που επιβάλλεται από τη διαδικασία της αποτίμησης είναι πιο κοντά στο προφίλ των τριάδων RDF. Οι βάσεις αυτές πράγματι έχουν δημιουργηθεί με κύριο στόχο τις συγκεκριμένες εφαρμογές. Από την άλλη πλευρά οι βάσεις SQL χαρακτηρίζονται από εντελώς διαφορετική φιλοσοφία σχεδίασης. Όμως πρέπει παράλληλα να υπολογίσουμε την επίπτωση που έχει ο διαφορετικός τρόπος χρήσης της μνήμης. Μπορεί και τα δύο συστήματα να καταλήγουν να χρησιμοποιούν την κύρια μνήμη αλλά υπάρχει μια μεγάλη διαφορά. Ο Fuseki το γνωρίζει και συμπεριφέρεται με πιο ταιριαστό τρόπο. Αντιθέτως, ο PostgreSQL χρησιμοποιεί τις ίδιες τεχνικές και βελτιώσεις που εφαρμόζει όταν τα δεδομένα βρίσκονται στον δίσκο, πράγμα που επηρεάζει τις συνολικές επιδόσεις της βάσης. Τέλος μια ακόμα σημαντική παράμετρος που μπορεί να επηρεάζει τις επιδόσεις είναι και το επιλεγμένο σχήμα της βάσης. Ίσως με διαφορετική οργάνωση της βάσης να υπήρχε κάποια βελτίωση. Το σίγουρο είναι ότι η διαφορά δεν είναι αμελητέα και πρέπει να ληφθεί υπόψη από τους χρήστες.

# 6

# *Συμπεράσματα*

Στην παρούσα εργασία υλοποιήθηκε μία εφαρμογή η οποία είναι σε θέση να αποτιμήσει τη μεταγραφή που δημιουργείται από το εργαλείο συλλογιστικής Rapid. Το Rapid αναπτύχθηκε και εξελίσσεται στο πλαίσιο του Εργαστηρίου Ψηφιακής Επεξεργασίας Εικόνας, Βίντεο και Πολυμέσων. Αρχικά, η υποστηριζόμενη περιγραφική λογική ήταν η DL-lite η οποία, λόγω περιορισμένης εκφραστικότητας, δίνει μεταγραφές που δεν απαιτούν πολύπλοκους μηχανισμούς για να αποτιμηθούν και να δώσουν τα τελικά αποτελέσματα. Οι μεταγραφές ερωτημάτων σε βάσεις γνώσεις DL-lite δίνονται ως σύνολα συζευκτικών ερωτημάτων, τα οποία είναι σε θέση να διαχειριστούν απευθείας οι σχεσιακές βάσεις δεδομένων. Ωστόσο, δεν συμβαίνει το ίδιο και στην περίπτωση της ELHI. Οι μεταγραφές ερωτημάτων όταν η εκφραστικότητα είναι σε αυτό το επίπεδο είναι αρκετά πιο πολύπλοκες. Για την αποτίμησή τους απαιτούνται βάσεις δεδομένων με επαγωγικά χαρακτηριστικά. Με αυτόν τον τρόπο η υποστήριξη της συγκεκριμένης περιγραφικής λογικής δημιούργησε την ανάγκη για έναν ισχυρότερο μηχανισμό αποτίμησης. Αυτή την ανάγκη ήρθε να καλύψει η εφαρμογή που υλοποιήθηκε στα πλαίσια αυτής της εργασίας.

Προκειμένου να καταστεί χρήσιμη στο διαρκώς μεταβαλλόμενο περιβάλλον του εργαστηρίου, η υλοποίηση έπρεπε να διακρίνεται από απλότητα στη σχεδίαση, ευκολία στην επέκταση και χαμηλές απαιτήσεις σε πόρους. Ο απλός και ξεκάθαρος σχεδιασμός επιτρέπει την εύκολη κατανόηση και εποπτεία των λειτουργιών των διαφόρων μερών του προγράμματος. Το γεγονός αυτό, σε συνδυασμό με την αναλυτική τεκμηρίωση των διαφόρων μεθόδων και πεδίων που περιλαμβάνει, διευκολύνει την ενσωμάτωσή του σε επόμενες

επεκτάσεις του Rapid ή ακόμα και σε κάποιο νέο εργαλείο. Για τους ίδιους λόγους, είναι εφικτή ακόμα και η χρήση τμημάτων του κώδικα που επιτελούν συγκεκριμένες επιθυμητές λειτουργίες. Η ευελιξία που επετεύχθη με τον αρθρωτό σχεδιασμό του στο κομμάτι της διαχείρισης βάσεων είναι ένα από τα σημαντικότερα χαρακτηριστικά του. Τα συστήματα αποθήκευσης που χρησιμοποιούνται σήμερα είναι πολύ πιθανό να αλλάξουν στο μέλλον ακολουθώντας τις ανάγκες της έρευνας. Αυτό δεν αποτρέπει τη συνέχιση της χρήσης της εφαρμογής. Η ύπαρξη της διεπαφής μέσω της οποίας γίνεται η επικοινωνία με τον χειριστή της βάσης, εγγυάται ότι η υποστήριξη για νέα συστήματα βάσεων δεδομένων μπορεί να προστεθεί με την ελάχιστη δυνατή προσπάθεια. Υπό αυτό το πρίσμα, μπορούμε να πούμε ότι οι στόχοι που είχαν τεθεί επετεύχθησαν μέσω αυτής της εφαρμογής.

Η υλοποιημένη εφαρμογή σε συνδυασμό με κάποια από τις υποστηριζόμενες βάσεις δεδομένων μπορούν να θεωρηθούν ως μία ενιαία οντότητα. Υπό αυτό το πρίσμα, ανακύπτει το ερώτημα της σύγκρισης με άλλα συστήματα επαγωγικών βάσεων δεδομένων. Υπάρχει πληθώρα συστημάτων αυτού του είδους, εμπορικού και μη χαρακτήρα, με τις πιθανότητες αύξησης του αριθμού τους να είναι μεγάλες. Ένα παράδειγμα είναι το RDFox, στο οποίο έγινε μια σύντομη αναφορά στο δεύτερο κεφάλαιο. Αναπτύχθηκε από το Πανεπιστήμιο της Οξφόρδης παράλληλα με το πρόγραμμα που παρουσιάστηκε στην παρούσα εργασία. Ένα γνωστό σύστημα που προϋπήρχε της ανάπτυξης της παρούσας εφαρμογής είναι το DLV. Οι επιδόσεις των εξειδικευμένων συστημάτων είναι ικανοποιητικότατες. Όμως η χρήση τέτοιων συστημάτων περιπλέκει την διαχείριση των συνόλων δεδομένων καθώς δεν μπορούν να συνεργαστούν απευθείας με τις υπάρχουσες βάσεις. Επιπλέον, οι απαιτήσεις της ανάπτυξης ενός νέου εργαλείου είναι ιδιαίτερες και δεν μπορούν να καλυφθούν πλήρως από μια έτοιμη εφαρμογή. Η πλήρης εποπτεία των παραμέτρων της αποτίμησης κάθε μεταγραφής, όπως για παράδειγμα ο αριθμός επαναλήψεων του βρόχου του αλγόριθμου αποτίμησης datalog, ο αριθμός των εκδοθέντων κανόνων και η δυνατότητα αποτελεσματικής τμηματοποίησής της είναι μεγάλης σημασίας για τον ποιοτικό χαρακτηρισμό της. Τα δεδομένα αυτά αποτελούν ένα χρήσιμο στοιχείο που δεν είναι δυνατό να προέλθει από τη χρήση τρίτων εφαρμογών.

## *6.1 Μελλοντικές επεκτάσεις*

Στη συγκεκριμένη περίπτωση, η ίδια η φύση της υλοποίησης καθοδηγεί την αναζήτηση για προτάσεις σχετικά με μελλοντικές επεκτάσεις. Η υποστήριξη και άλλων συστημάτων διαχείρισης βάσεων δεδομένων αυξάνει σημαντικά την χρησιμότητα της εφαρμογής. Ένα

καλό σημείο εκκίνησης για κάποιον που ενδιαφέρεται να επεκτείνει την εφαρμογή είναι η δημιουργία ενός αρθρώματος που θα υποστηρίζει ένα διαφορετικό σύστημα διαχείρισης σχεσιακών βάσεων δεδομένων. Οι ομοιότητες που υπάρχουν μεταξύ αυτών των συστημάτων καθιστά πολύ εύκολη την επέκταση σε συστήματα άλλων κατασκευαστών. Οι υπάρχουσες κλάσεις για την υποστήριξη του PostgreSQL μπορούν να αποτελέσουν έναν ιδανικό οδηγό. Πέρα από την υποστήριξη περισσότερων βάσεων δεδομένων, θα μπορούσε να δημιουργηθεί ένα πολύ γρήγορο, εξειδικευμένο για κατηγορήματα μέχρι δύο όρων, σύστημα αποθήκευσης στην κύρια μνήμη. Μια τέτοια επέκταση θα επέτρεπε την άμεση αποτίμηση μεταγραφών χωρίς να απαιτείται η εγκατάσταση κάποιου εξωτερικού συστήματος βάσης δεδομένων. Σε αυτή την περίπτωση, βέβαια, το μέγεθος των συνόλων δεδομένων περιορίζεται από το μέγεθος της κύριας μνήμης του υπολογιστή που χρησιμοποιείται. Σε συνδυασμό με το σύστημα αποθήκευσης στην κύρια μνήμη, θα μπορούσε να αναπτυχθεί ένας διαχειριστής βάσης που θα υποστηρίζει λειτουργίες ανάκτησης δεδομένων από βάση. Με αυτό τον τρόπο, θα μπορούν να απαντώνται ερωτήματα σε βάσεις όπου δεν επιτρέπεται η εκτέλεση εντολών ενημέρωσης με τρόπο διαφανή προς τον χρήστη της εφαρμογής.

# 7

# *Βιβλιογραφία*


[1] F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, και P. F. Patel-Schneider, Επιμ., *The Description Logic Handbook: Theory, Implementation, and Applications*, 2η έκδ. Cambridge University Press, 2010.

[2] E. Mendelsohn, *Introduction to Mathematical Logic*, 3η έκδ. Belmont, CA, USA: Wadsworth Publ. Co., 1987.

[3] R. Quillian, 'Word concepts: A theory and simulation of some basic semantic capabilities.', *Behav. Sci.*, τ. 12, τχ. 5, σσ 410–430, 1967.

[4] R. Brachman και H. Levesque, 'Object-Oriented Representation', στο *Knowledge Representation and Reasoning*, 1η έκδ., Elsevier, 2004, σσ 135–154.

[5] R. Brachman και H. Levesque, 'Rules in Production Systems', στο *Knowledge Representation and Reasoning*, 1η έκδ., Elsevier, 2004, σσ 117–134.

[6] U. Sattler, 'A concept language extended with different kinds of transitive roles', στο *20th Annual German Conference on Artificial Intelligence: Advances in Artificial Intelligence*, 1996, σσ 333–345.

[7] A. Artale και E. Franconi, 'Introducing temporal description logics', στο *Sixth International Workshop on Temporal Representation and Reasoning. TIME-99*, 1999, σσ 2–5.

[8] F. Baader, R. Kusters, και F. Wolter, 'Extensions to description logics', στο *The Description Logic Handbook: Theory, Implementation and Applications*, F. Baader, D.

Calvanese, D. L. McGuinness, D. Nardi, και P. F. Patel-Schneider, Επιμ. New York, NY, USA: Cambridge University Press, 2003, σσ 219–261.

[9] R. Reiter, 'On Closed World Data Bases', στο *Logic and Data Bases*, H. Gallaire και J. Minker, Επιμ. Boston, MA: Springer US, 1978, σσ 55–76.

[10] A. Artale, D. Calvanese, R. Kontchakov, και M. Zakharyaschev, 'The DL-Lite family and relations', *J. Artif. Intell. Res.*, τ. 36, σσ 1–69, 2009.

[11] F. Baader, I. Horrocks, και U. Sattler, 'Description Logics', στο *Handbook on Ontologies*, 2η έκδ., S. Staab και R. Studer, Επιμ. Springer, 2009, σσ 21–43.

[12] F. Baader, I. Horrocks, και U. Sattler, 'Description Logics', στο *Handbook of Knowledge Representation*, F. van Harmelen, V. Lifschitz, και B. Porter, Επιμ. Elsevier, 2008, σσ 135–180.

[13] T. Eiter, G. Ianni, T. Lukasiewicz, R. Schindlauer, και H. Tompits, 'Combining Answer Set Programming with Description Logics for the Semantic Web', *Artif. Intell.*, τ. 172, τχ. 12–13, σσ 1495–1539, 2008.

[14] B. Motik, U. Sattler, και R. Studer, 'Query Answering for OWL-DL with Rules', *Web Semant. Sci. Serv. Agents World Wide Web*, τ. 3, τχ. 1, σσ 41–60, 2005.

[15] F. M. Donini, M. Lenzerini, D. Nardi, και A. Schaerf, 'AL-log: Integrating Datalog and Description Logics', *J. Intell. Inf. Syst.*, τ. 10, τχ. 3, σσ 227–252, 1998.

[16] A. Y. Levy και M.-C. Rousset, 'CARIN : A Representation Language Combining Horn rules and Description Logics', *Artif. Intell.*, τ. 104, τχ. 1–2, σσ 165–209, 1998.

[17] A. Poggi, D. Lembo, D. Calvanese, G. De Giacomo, M. Lenzerini, και R. Rosati, 'Linking Data to Ontologies', στο *Journal on Data Semantics X*, S. Spaccapietra, Επιμ. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, σσ 133–173.

[18] S. Tobies, 'Complexity Results and Practical Algorithms for Logics in Knowledge Representation', RWTH Aachen, 2001.

[19] R. Brachman και H. Levesque, 'The Tradeoff between Expressiveness and Tractability', στο *Knowledge Representation and Reasoning*, 1η έκδ., Elsevier, 2004, σσ 327–342.

[20] F. Baader, S. Brandt, και C. Lutz, 'Pushing the EL envelope', στο *International Joint Conference on Artificial Intelligence*, 2005, σσ 364–369.

[21] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, και R. Rosati, 'DL-Lite: Tractable description logics for ontologies', στο *20th Nat. Conf. on Artificial Intelligence (AAAI 2005)*, 2005, τ. 5, σσ 602–607.

[22] B. Nebel, 'Terminological reasoning is inherently intractable', *Artif. Intell.*, τ. 43, τχ.

2, σσ 235–249, 1990.

[23] Y. Kazakov και H. de Nivelle, ‘Subsumption of concepts in FL0 for (cyclic) terminologies with respect to descriptive semantics is PSPACE-complete’, στο *DL 03*, 2003.

[24] M. Hofmann, ‘Proof-Theoretic Approach to Description-Logic’, στο *Proceedings of the 20th Annual IEEE Symposium on Logic in Computer Science*, 2005, σσ 229–237.

[25] S. Brandt, ‘Polynomial Time Reasoning in a Description Logic with Existential Restrictions, GCI Axioms, and---What else?’, στο *Proceedings of the 16th European Conference on Artificial Intelligence*, 2004, σσ 298–302.

[26] D. Trivela, G. Stoilos, A. Chortaras, και G. Stamou, ‘Optimising resolution-based rewriting algorithms for OWL ontologies’, *J. Web Semant.*, τ. 33, σσ 30–49, 2015.

[27] H. Pérez-Urbina, B. Motik, και I. Horrocks, ‘Tractable query answering and rewriting under description logic constraints’, *J. Appl. Log.*, τ. 8, τχ. 2, σσ 186–209, 2010.

[28] A. Krisnadhi και C. Lutz, ‘Data Complexity in the EL family of DLs’, στο *International Workshop on Description Logics (DL2007)*, 2007.

[29] S. S. A. Silberschatz, H. Korth, *Συστήματα Βάσεων Δεδομένων*, 4η έκδ. Αθήνα: M. Γκιούρδας, 2004.

[30] J. Rumbaugh, I. Jacobson, και G. Booch, *Unified Modeling Language Reference Manual, The*, 2η έκδ. Pearson Higher Education, 2004.

[31] W3C OWL Working Group, ‘OWL 2 Web Ontology Language Document Overview’, *W3C*, 2012. [Έκδοση σε ψηφιακή μορφή]. Διαθέσιμο στο: http://www.w3.org/TR/owl2-overview/.

[32] K. Dentler, R. Cornet, A. Ten Teije, και N. De Keizer, ‘Comparison of reasoners for large ontologies in the OWL 2 EL profile’, *Semant. Web*, τ. 2, τχ. 2, σσ 71–87, 2011.

[33] M. Kaminski, Y. Nenov, και B. C. Grau, ‘Datalog Rewritability of Disjunctive Datalog Programs and its Applications to Ontology Reasoning’, στο *AAAI Conference on Artificial Intelligence*, 2014, σσ 1077–1083.

[34] I. Horrocks, P. F. Patel-Schneider, και F. Van Harmelen, ‘From SHIQ and RDF to OWL: The Making of a Web Ontology Language’, *J. Web Semant.*, τ. 1, τχ. 1, σσ 7–26, 2003.

[35] F. Gandon και G. Schreiber, ‘RDF 1.1 XML Syntax’, *W3C*, 2014. [Έκδοση σε ψηφιακή μορφή]. Διαθέσιμο στο: https://www.w3.org/TR/2014/REC-rdf-syntax-grammar-20140225/.

[36] F. M. Donini, M. Lenzerini, D. Nardi, και A. Schaerf, ‘Reasoning in Description

Logics', στο *Principles of Knowledge representation*, τ. 1, G. Brewka, Επιμ. 1996, σσ 191–236.

[37] J. Bock, P. Haase, Q. Ji, και R. Volz, 'Benchmarking OWL reasoners', στο *Workshop on Advancing Reasoning on the Web: Scalability and Commonsense*, 2008.

[38] F. Baader και U. Sattler, 'An Overview of Tableau Algorithms for Description Logics', *Stud. Log.*, τ. 69, τχ. 1, σσ 5–40, 2001.

[39] P. Bresciani, E. Franconi, και S. Tessaris, 'Implementing and Testing Expressive Description Logics: a Preliminary Report', στο *Description Logics Workshop 1995 (DL'95)*, 1995, σσ 131–139.

[40] B. Hollunder, W. Nutt, και M. Schmidt-Schauß, 'Subsumption algorithms for concept description languages', στο *9th Eur. Conf. on Artificial Intelligence (ECAI'90)*, 1990, σσ 348–353.

[41] R. Möller και V. Haarslev, 'Tableau-Based Reasoning', στο *Handbook on Ontologies*, 2η έκδ., S. Staab και R. Studer, Επιμ. Berlin: Springer-Verlag, 2009, σσ 509–512.

[42] B. Motik και U. Sattler, 'A comparison of reasoning techniques for querying large description logic aboxes', στο *Proceedings of the 13th international conference on Logic for Programming*, 2006, σσ 227–241.

[43] U. Hustadt και B. Motik, 'Description logics and disjunctive datalog : The story so far', στο *CEUR Workshop Proceedings*, 2005, τ. 147.

[44] I. R. Horrocks, 'Implementation and Optimization Techniques', στο *The Description Logic Handbook: Theory, Implementation and Applications*, Cambridge: Cambridge University Press, 2003, σσ 306–346.

[45] B. Motik, R. Shearer, και I. Horrocks, 'Hypertableau reasoning for description logics', *J. Artif. Intell. Res.*, τ. 36, τχ. 1, σσ 165–228, 2009.

[46] I. Horrocks, 'Using an Expressive Description Logic: FaCT or Fiction?', στο *Principles of Knowledge Representation and Reasoning: Proceedings of the Sixth International Conference (KR'98)*, 1998, σσ 636–647.

[47] D. Tsarkov και I. Horrocks, 'FaCT++ Description Logic Reasoner: System Description', στο *International Joint Conference on Automated Reasoning (IJCAR)*, 2006, σσ 292–297.

[48] I. Horrocks, U. Sattler, και S. Tobies, 'Reasoning with Individuals for the Description Logic SHIQ', στο *Proceedings of the 17th International Conference on Automated Deduction*, 2000, σσ 482–496.

[49] V. Haarslev, R. Möller, και M. Wessel, 'Querying the semantic web with racer+ nrql', στο *KI-2004 Workshop on Applications of Description Logics (ADL'04)*, 2004.

[50] V. Haarslev και R. Möller, ‘Expressive ABox Reasoning with Number Restrictions, Role Hierarchies, and Transitively Closed Roles’, στο *International Conference on Principles of Knowledge Representation and Reasoning (KR’2000)*, 2000, σσ 273–284.

[51] V. Haarslev και R. Müller, ‘RACER System Description’, στο *1st International Joint Conference on Automated Reasoning*, 2001, τ. 2083, σσ 701–706.

[52] T. Liebig, ‘Reasoning with OWL - System Support and Insights’, Ulm, Germany, 2006.

[53] R. Shearer, B. Motik, και I. Horrocks, ‘HermiT: A Highly-Efficient OWL Reasoner.’, στο *Workshop on OWL: Experiences and Directions*, 2008.

[54] E. Sirin και B. Parsia, ‘Pellet system description’, στο *Workshop on Description Logics, DL*, 2006, τ. 189.

[55] E. Sirin, B. Parsia, B. C. Grau, A. Kalyanpur, και Y. Katz, ‘Pellet: A practical OWL-DL reasoner’, *Web Semant. Sci. Serv. Agents World Wide Web*, τ. 5, τχ. 2, σσ 51–53, 2007.

[56] J. Broekstra, A. Kampman, και F. van Harmelen, ‘Sesame : A generic architecture for storing and querying RDF and RDF Schema’, στο *International Semantic Web Conference*, 2002, σσ 54–68.

[57] B. Bishop, A. Kiryakov, D. Ognyanoff, I. Peikov, Z. Tashev, και R. Velkov, ‘OWLIM: A family of scalable semantic repositories’, *Semant. Web*, τ. 2, τχ. 1, σσ 33–42, 2011.

[58] H. J. ter Horst, ‘Combining RDF and Part of OWL with Rules: Semantics, Decidability, Complexity’, στο *Proceedings of the 4th International Conference on The Semantic Web*, 2005, σσ 668–684.

[59] G. Meditskos και N. Bassiliades, ‘A rule-based object-oriented OWL reasoner’, *IEEE Trans. Knowl. Data Eng.*, τ. 20, τχ. 3, σσ 397–410, 2008.

[60] A. Chniti, S. Dehors, P. Albert, και J. Charlet, ‘Authoring Business Rules Grounded in OWL Ontologies’, στο *RULEML*, 2010, σσ 297–304.

[61] C. Forgy, ‘Rete: A Fast Algorithm for the Many Patterns/Many Objects Match Problem’, *Artif. Intell.*, τ. 19, τχ. 1, σσ 17–37, 1982.

[62] M. Kifer, G. Lausen, και J. Wu, ‘Logical foundations of object-oriented and frame-based languages’, *J. ACM*, τ. 42, τχ. 4, σσ 741–843, 1995.

[63] G. Yang, M. Kifer, και C. Zhao, ‘Flora-2: A Rule-Based Knowledge Representation and Inference Infrastructure for the Semantic Web’, στο *International Conference of Ontologies Databases and Applications of Semantics*, 2003, σσ 1–18.

[64] K. Sagonas, T. Swift, και D. S. Warren, ‘XSB as an Efficient Deductive Database Engine’, στο *Int. Conf. on the Management of Data*, 1994, σσ 442–453.

[65] T. Swift και D. S. Warren, ‘XSB: Extending Prolog with Tabled Logic Programming’, *Program. Theory Pract. Log. Program.*, τ. 12, τχ. Special Issue 1-2, σσ 157–187, 2012.

[66] B. Ludaescher, R. Himmeroeder, G. Lausen, W. May, και C. Schlepphorst, ‘Managing semistructured data with FLORID: A deductive object-oriented perspective’, *Inf. Syst.*, τ. 23, τχ. 8, σσ 589–613, 1998.

[67] K. Wilkinson, C. Sayers, H. Kuno, και D. Reynolds, ‘Efficient RDF storage and retrieval in Jena2’, στο *International Workshop on Semantic Web and Databases*, 2003, σσ 120–139.

[68] Y. Nenov, R. Piro, B. Motik, I. Horrocks, Z. Wu, και J. Banerjee, ‘RDFox: A Highly-Scalable RDF Store’, στο *The Semantic Web - ISWC 2015: 14th International Semantic Web Conference, Proceedings Part II*, 2015, σσ 3–20.

[69] D. Tsarkov, A. Riazanov, S. Bechhofer, και I. Horrocks, ‘Using vampire to reason with OWL’, στο *LNCS-ISWC*, 2004, σσ 471–485.

[70] A. Riazanov και A. Voronkov, ‘The Design and Implementation of Vampire’, *AI Commun.*, τ. 15, τχ. 2, σσ 91–110, 2002.

[71] D. Calvanese, G. Giacomo, D. Lembo, M. Lenzerini, και R. Rosati, ‘Tractable Reasoning and Efficient Query Answering in Description Logics: The DL-Lite Family’, *J. Autom. Reason.*, τ. 39, τχ. 3, σσ 385–429, 2007.

[72] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, και R. Rosati, ‘Data Complexity of Query Answering in Description Logics’, στο *10th Int. Conf. on the Principles of Knowledge Representation and Reasoning*, 2006, τ. 6, σσ 260–270.

[73] U. Hustadt, B. Motik, και U. Sattler, ‘Data complexity of reasoning in very expressive description logics’, στο *International Joint Conference on Artificial Intelligence*, 2005, σσ 466–471.

[74] T. Eiter, G. Gottlob, και H. Mannila, ‘Disjunctive Datalog’, *ACM Trans. Database Syst.*, τ. 22, τχ. 3, σσ 364–418, 1997.

[75] S. Kikot, R. Kontchakov, V. Podolskii, και M. Zakharyaschev, ‘Exponential lower bounds and separation for query rewriting’, στο *International Colloquium on Automata, Languages, and Programming*, 2012, σσ 263–274.

[76] R. Rosati και A. Almatelli, ‘Improving Query Answering over DL-Lite Ontologies’, στο *Conference on Principles of Knowledge Representation and Reasoning*, 2010, σσ 290–300.

[77] R. Rosati, ‘Query Rewriting under Extensional Constraints in DL-Lite’, στο *25th International Workshop on Description Logics*, 2012, σσ 323–334.

[78] A. Acciarri, D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, M. Palmieri, και R. Rosati, ‘QuOnto: Querying Ontologies’, στο *20th National Conference on Artificial Intelligence (AAAI)*, 2005, σσ 1670–1671.

[79] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, A. Poggi, M. Rodriguez-Muro, R. Rosati, M. Ruzzi, και D. F. Savo, ‘The MASTRO system for ontology-based data access’, *Semant. Web*, τ. 2, τχ. 1, σσ 43–53, 2011.

[80] M. Rodriguez-Muro, R. Kontchakov, και M. Zakharyaschev, ‘Query Rewriting and Optimisation with Database Dependencies in Ontop’, στο *Informal Proceedings of the 26th International Workshop on Description Logics*, 2013, σσ 917–929.

[81] T. Venetis, G. Stoilos, και G. Stamou, ‘Query Extensions and Incremental Query Rewriting for OWL 2 QL Ontologies’, *J. Data Semant.*, τ. 3, τχ. 1, σσ 1–23, 2014.

[82] M. Rodrıguez-Muro και D. Calvanese, ‘Quest, an OWL 2 QL reasoner for ontology-based data access’, στο *9th Int. Workshop on OWL: Experiments and Directions*, 2012.

[83] G. Gottlob, G. Orsi, και A. Pieris, ‘Ontological queries: Rewriting and optimization’, στο *International Conference on Data Engineering*, 2011, σσ 2–13.

[84] H. Pérez-Urbina, B. Motik, και I. Horrocks, ‘A comparison of query rewriting techniques for DL-lite’, στο *Description Logics*, 2009, τ. 477, σ 29.

[85] T. Eiter, M. Ortiz, M. Simkus, T. Tran, και G. Xiao, ‘Query Rewriting for Horn-SHIQ Plus Rules.’, στο *Aaai*, 2012, σσ 726–733.

[86] I. Horrocks, P. F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, και M. Dean, ‘SWRL: A Semantic Web Rule Language Combining OWL and RuleML’, *W3C*, 2004. [Έκδοση σε ψηφιακή μορφή]. Διαθέσιμο στο: https://www.w3.org/Submission/2004/SUBM-SWRL-20040521/.

[87] S. Decker, M. Erdmann, D. Fensel, και R. Studer, ‘Ontobroker: Ontology based access to distributed and semi-structured information’, στο *Database Semantics IFIP — The International Federation for Information Processing Volume 11*, 1999, σσ 351–369.

[88] G. Lukácsy και P. Szeredi, ‘Efficient description logic reasoning in Prolog: The DLog system’, *Theory Pract. Log. Program.*, τ. 9, τχ. 3, σσ 343–414, 2009.

[89] J. W. Lloyd, *Foundations of Logic Programming*, 2η εκτ. έκ. New York: Springer - Verlag, 1987.

[90] C. Green και B. Raphael, ‘The use of theorem-proving techniques in question-answering systems’, στο *ACM ’68 Proceedings of the 1968 23rd ACM national*

*conference*, 1968, σσ 169–181.

[91] R. Kowalski, ‘Predicate Logic as a Programming Language’, στο *Information Processing ’74*, 1974, σσ 569–574.

[92] A. Yaacoub και A. Awada, ‘Information Flow in Datalog using Very Naive and Semi Naive Bottom-Up Evaluation Techiques’, *Int. J. Comput. Sci. Theory Appl.*, τ. 5, τχ. 2, 2016.

[93] L. Sterling και E. Y. Shapiro, *The Art of Prolog*, 2η έκδ. Cambridge, Massachusetts: MIT Press, 1994.

[94] R. Brachman και H. Levesque, *Knowledge Representation and Reasoning*, 1η έκδ. Elsevier, 2004.

[95] E. Dantsin, T. Eiter, G. Gottlob, και A. Voronkov, ‘Complexity and expressive power of logic programming’, *ACM Comput. Surv.*, τ. 33, τχ. 3, σσ 374–425, 2001.

[96] C. Chang και H. Keiser, *Model Theory*, 3η έκδ. Mineola, New York: Dover, 2013.

[97] S. Abiteboul, R. Hull, και V. Vianu, *Foundations of Databases*, 1η έκδ. Boston, MA, USA: Addison-Wesley, 1995.

[98] M. Van Emdem και R. Kowalski, ‘The Semantics of Predicate Logic as a Programming Language’, *JACM*, τ. 23, τχ. 4, σσ 733–742, 1976.

[99] D. Hilbert, ‘Axiomatisches Denken’, *Math. Ann.*, τ. 78, σσ 405–415, 1918.

[100] J. Robinson, ‘A Machine-Oriented Logic Based on the Resolution-Principle.’, *JACM*, τ. 12, τχ. 1, σσ 23–41, 1965.

[101] R. Kowalski και D. Kuehner, ‘Linear resolution with selection function’, *Artificial Intell.*, τ. 2, σσ 227–260, 1971.

[102] K. Apt και M. van Emden, ‘Contributions to the Theory of Logic Programming’, *JACM*, τ. 29, τχ. 3, σσ 841–862, 1982.

[103] F. Bry, N. Eisinger, T. Eiter, T. Furche, G. Gottlob, C. Ley, B. Linse, R. Pichler, και F. Wei, ‘Foundations of Rule-based Query Answering’, στο *Proceedings of the Third International Summer School Conference on Reasoning Web*, 2007, σσ 1–153.

[104] L. Vielle, ‘A database complete proof procedure based on SLD resolution’, στο *4th International Conference on Logic Programming*, 1987, σσ 74–103.

[105] S. Ceri, G. Gotlob, και L. Tanca, ‘What You Always Wanted to Know About Datalog ( And Never Dared to Ask )’, *IEEE Trans. Knowl. Data Eng.*, τ. 1, τχ. 1, σσ 146–166, 1989.

[106] R. Tarjan, ‘Depth-first search and linear graph algorithms’, στο *12th Annual Symposium on Switching and Automata Theory (swat 1971)*, 1971, τ. 1, τχ. 2, σσ 146–

160.

[107] G. Schreiber και Y. Raimond, ‘RDF 1.1 Primer’, *W3C*, 2014. [Έκδοση σε ψηφιακή μορφή]. Διαθέσιμο στο: https://www.w3.org/TR/2014/NOTE-rdf11-primer-20140624/.

[108] S. Alexaki, V. Christophides, G. Karvounarakis, D. Plexousakis, και K. Tolle, ‘The RDFSuite: Managing Voluminous RDF Description Bases’, στο *International Workshop on the Semantic Web*, 2001, σσ 1–13.

[109] S. Harris και N. Gibbins, ‘3store: Efficient Bulk RDF Storage’, στο *Proceedings of the 1st International Workshop on Practical and Scalable Semantic Systems (PSSS’03)*, 2003, σσ 1–20.

[110] S. Harris και A. Seaborne, ‘SPARQL 1.1 Query Language’, *W3C*, 2013. [Έκδοση σε ψηφιακή μορφή]. Διαθέσιμο στο: https://www.w3.org/TR/2013/REC-sparql11-query-20130321/.

[111] P. Gearon, A. Passant, και A. Polleres, ‘SPARQL 1.1 Update’, *W3C*, 2013. [Έκδοση σε ψηφιακή μορφή]. Διαθέσιμο στο: https://www.w3.org/TR/2013/REC-sparql11-update-20130321/.

[112] K. Schild, ‘A correspondence theory for terminological logics: preliminary report’, στο *12th International Joint Conference on Artificial Intelligence*, 1991, σσ 466–471.