ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

Ανάπτυξη Μηχανής Συλλογιστικής για την Περιγραφική Λογική *fuzzy-EL*$^+$

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Μιχαήλ Α. Μακαρονίδης

Επιβλέπων: Γεώργιος Στάμου
Λέκτορας Ε.Μ.Π.

Αθήνα, Ιούλιος 2009

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

Ανάπτυξη Μηχανής Συλλογιστικής για την Περιγραφική Λογική *fuzzy-EL*$^+$

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Μιχαήλ Α. Μακαρονίδης

Επιβλέπων: Γεώργιος Στάμου
Λέκτορας Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 22^{α} Ιουλίου 2009.

.....	.....	.....
Σ. Κόλλιας	Γ. Στάμου	Α. Σταφυλοπάτης
Καθηγητής Ε.Μ.Π.	Λέκτορας Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.

Αθήνα, Ιούλιος 2009

.....

Μιχαήλ Α. Μακαρονίδης
Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

1. Περίληψη

Σκοπός της παρούσας Διπλωματικής Εργασίας είναι η ανάπτυξη μίας μηχανής συλλογιστικής για τις Περιγραφικές Λογικές $\mathcal{EL}^+$ και fuzzy-$\mathcal{EL}^+$. Η βασική υπηρεσία συλλογιστικής που παρέχεται για τις Περιγραφικές Λογικές αυτές είναι εκείνη της ταξινόμησης (classification), δηλαδή του υπολογισμού της πλήρους ιεραρχίας υπαγωγών μεταξύ όλων των ονοματικών εννοιών που εμφανίζονται σε μία οντολογία. Ο αλγόριθμος συλλογιστικής που υλοποιήθηκε ανήκει στην οικογένεια των αλγορίθμων δομικής υπαγωγής και χαρακτηρίζεται από πολυωνυμική χρονική πολυπλοκότητα.

Βασικός στόχος της υλοποίησης είναι η τελική ταχύτητα του συστήματος να είναι συγκρίσιμη με εξειδικευμένες μηχανές συλλογιστικής για σαφείς Περιγραφικές Λογικές της οικογένειας των $\mathcal{EL}$, όπως η μηχανή συλλογιστικής CEL. Για την επίτευξη αυτού του στόχου, σημαντικότατη ήταν η εισαγωγή πολυάριθμων βελτιστοποιήσεων στον θεωρητικό αλγόριθμο συλλογιστικής, όπως αυτές παρουσιάζονται στη βιβλιογραφία. Επίσης, χρησιμοποιήθηκαν διάφορες τεχνικές περιορισμού του χώρου αναζήτησης προκειμένου να αυξηθεί η ταχύτητα απόκρισης του συστήματος, καθώς και σημαντικές τεχνικές μείωσης των χωρικών απαιτήσεων του αλγορίθμου. Τέλος, εισήχθησαν μέθοδοι ελαχιστοποίησης του χρονικού κόστους δαπανηρών διαδικασιών υπολογισμού με χρήση τεχνικών αποθήκευσης των αποτελεσμάτων, αντίστοιχα με τις κρυφές μνήμες.

Ένας δεύτερος στόχος της παρούσας υλοποίησης είναι η ανάπτυξη των καταλλήλων διεπαφών και τρόπων επικοινωνίας, έτσι ώστε η μηχανή συλλογιστικής να μπορεί εύκολα να χρησιμοποιηθεί από άλλα έργα λογισμικού ή να ενσωματωθεί σε αυτά. Για τον λόγο αυτό υλοποιήθηκαν πολλές διαφορετικές μέθοδοι εκτέλεσης ερωτημάτων και εξαγωγής των αποτελεσμάτων, καθώς και δυνατότητα απομακρυσμένης εκτέλεσης ερωτημάτων.

1.1 Λέξεις Κλειδιά

Συλλογιστική, Μηχανή Συλλογιστικής, Αλγόριθμος Συλλογιστικής, Περιγραφική Λογική, Ασαφής Περιγραφική Λογική, EL, Fuzzy EL, Ταξινόμηση, Δομική Υπαγωγή, Υπαγωγή, Πολυωνυμικός Αλγόριθμος, Βελτιστοποιήσεις Αλγορίθμων Συλλογιστικής, CEL.

2. Abstract

The purpose of this Diploma Thesis is the development of a reasoner for the Description Logics $\mathcal{EL}^+$ and fuzzy-$\mathcal{EL}^+$. The main reasoning service offered for these Description Logics is classification, i.e., the computation of the complete subsumption hierarchy between all concept names occurring in an ontology. The reasoning algorithm employed is based on structural subsumption algorithms and exhibits polynomial time-complexity.

The main goal is the classification speed of the final system to be comparable to the one of specialized reasoners for crisp Description Logics of the $\mathcal{EL}$ family, such as CEL. For the achievement of such a goal, great was the importance of the introduction of numerous optimizations for the theoretical reasoning algorithm, as those presented on related bibliographical sources. Furthermore, many techniques were used to reduce the search space in order to increase the classification speed and to reduce the algorithm's space requirements. Finally, methods such as caching were introduced in order to minimize the time required for certain complex series of calculations.

A different goal for the development of the reasoning system is the support of appropriate interfaces and communication methods, such as those required in order to use or embed the reasoner in other software projects. As a result, the developed reasoner supports many different methods for querying information and for result extraction, and can even provide remote reasoning services, therefore enabling its use as a back-end reasoner.

2.1 Keywords

Reasoning, Reasoner, Reasoning Algorithm, Description Logic, Fuzzy Description Logic, EL, Fuzzy EL, Classification, Structural Subsumption, Subsumption, Polynomial Algorithm, Reasoning Algorithm Optimizations, CEL.

Κατάλογος περιεχομένων

3. Εισαγωγή

3.1 Γενικά

Η καταγραφή της ανθρώπινης γνώσης σε ένα Υπολογιστικό Σύστημα ή αλγόριθμο αποτελεί βασικό πρόβλημα το οποίο καλείται να επιλύσει ο τομέας της Επιστήμης των Υπολογιστών και ιδιαίτερα ο τομέας της Τεχνητής Νοημοσύνης. Μετά την καταγραφή της γνώσης στο σύστημα, αυτή θα μπορεί να αξιοποιηθεί για την εξαγωγή συμπερασμάτων και τη διενέργεια συλλογισμών, με ποιότητα ιδανικά αντίστοιχη της ανθρώπινης ικανότητας συλλογιστικής και σκέψης. Το πρόβλημα αυτό ξεκίνησε να απασχολεί τον άνθρωπο πολύ πριν την εμφάνιση των πρώτων υπολογιστών: ήδη από την εποχή του Αριστοτέλη γινόταν προσπάθεια να βρεθεί μία τυπική (formal) μέθοδος καταγραφής της ανθρώπινης γνώσης. Η φυσική γλώσσα που χρησιμοποιεί ο άνθρωπος παρουσιάζει αμφισημία – επινοήθηκαν λοιπόν ειδικές "γλώσσες" περιγραφής γνώσης, οι οποίες αναφέρονται ως **Γλώσσες Αναπαράστασης Γνώσης (Knowledge Representation Languages)**. Ορισμένες από της γλώσσες αναπαράστασης γνώσης είναι η **Προτασιακή Λογική (Propositional Logic)**, η **Κατηγορηματική Λογική Πρώτης Τάξης (First-Order Predicate Logic)**, τα **Σημασιολογικά Δίκτυα (Semantic Networks)** και οι **Περιγραφικές Λογικές (Description Logics)**, τις οποίες και αφορά η παρούσα διπλωματική εργασία. Εκτός από τα τυπικά τους χαρακτηριστικά (αλφάβητο, συντακτικό, σημασιολογία), οι γλώσσες αναπαράστασης γνώσης χαρακτηρίζονται και από ένα επιπλέον σημαντικό στοιχείο: το μηχανισμό εξαγωγής συμπερασμάτων ή **αλγόριθμο συλλογιστικής (reasoning alogorithm)**. Η σημασία του αλγορίθμου συλλογιστικής οφείλεται στο γεγονός ότι αυτός αυτό δίδει τη δυνατότητα παραγωγής νέας γνώσης από την ήδη υπάρχουσα ([1]).

Ο αρχικός στόχος της εύρεσης ενός συστήματος Περιγραφικών Λογικών που από τη μία πλευρά θα επέτρεπε ορθή και πλήρη συλλογιστική με αλγόριθμο πολυωνυμικού χρόνου, ενώ από την άλλη πλευρά θα παρείχε την εκφραστική ικανότητα που απαιτείται για να είναι δυνατή η χρήση του σε πραγματικές εφαρμογές, γρήγορα εγκαταλείφθηκε ως ανέφικτος. Αυτό οφειλόταν κυρίως σε δύο λόγους:

1. Κατά πρώτον στη μη βατή, δηλαδή μη πολυωνυμική, πολυπλοκότητα του αλγορίθμου συλλογιστικής ακόμα και σε χαμηλής εκφραστικότητας περιγραφικές λογικές, ιδίως εξ' αιτίας της παρουσίας των **Σωμάτων Ορολογίας (Terminological Box, Tbox)**. Δύο σημαντικές πηγές πολυπλοκότητας είναι:

   i. **Το OR-Branching:** Η ύπαρξη του τελεστή της ένωσης οδηγεί στην αναζήτηση εκθετικά πολλών διαφορετικών μοντέλων που μπορεί να ικανοποιούν μία έννοια. Το πρόβλημα αυτό εμφανίζεται και στις περιπτώσεις όπου απαιτείται συλλογιστική με συνθήκες, όπως για παράδειγμα όταν η Περιγραφική Λογική υποστηρίζει υπαρξιακούς ποσοδείκτες και περιορισμούς πληθικότητας.

   ii. **Το AND-Branching:** Το πρόβλημα αυτό προέρχεται από την ταυτόχρονη υποστήριξη υπαρξιακών και καθολικών ποσοδεικτών. Στην περίπτωση αυτή τα πιθανά διαφορετικά μοντέλα μπορεί να είναι εκθετικά σε μέγεθος, με αποτέλεσμα να απαιτείται ο έλεγχος εκθετικά πολλών πιθανών αντιφάσεων. Βέβαια, αυτό μπορεί να αντιμετωπιστεί σε ορισμένες περιπτώσεις, όπως στην $\mathcal{ALE}$.

2. Κατά δεύτερον στο γεγονός ότι οι περισσότερες εφαρμογές που εξετάζονταν τότε απαιτούσαν τη χρήση γλωσσών ακόμα υψηλότερης εκφραστικότητας.

Παρ' όλα αυτά, πραγματικές εφαρμογές αυτές καταστήθησαν εφικτές καθώς υλοποιήθηκαν **μηχανές συλλογιστικής (reasoners)** με διάφορες βελτιστοποιήσεις – συνήθως βασιζόμενες σε αλγορίθμους tableau –, οι οποίες παρείχαν καλή συμπεριφορά στην πράξη, παρά το γεγονός ότι η θεωρητική τους πολυπλοκότητα στη χειρότερη περίπτωση παρέμενε πολύ υψηλή ([2]).

Στη σημερινή εποχή, οι πρόσφατες εξελίξεις στην οικογένεια των περιγραφικών λογικών $\mathcal{EL}$ έχουν συντελέσει στη μερική πραγμάτωση αυτού του αρχικού στόχου. Έχει αποδειχθεί θεωρητικά πως η συλλογιστική στην $\mathcal{EL}$ και σε επεκτάσεις της παραμένει βατή (πολυωνυμικού χρόνου) ακόμα και παρουσία Σωμάτων Ορολογίας (TBoxes) και **Γενικευμένων Αξιωμάτων Υπαγωγής (General Concept Inclusions – GCIs)** ([3]). Επίσης, παρά τη χαμηλή εκφραστικότητα της οικογένειας $\mathcal{EL}$ συγκριτικά με άλλες Περιγραφικές Λογικές, αυτή είναι κατάλληλη για έναν αριθμό από σημαντικές εφαρμογές, ιδίως στον τομέα της επιστήμης της βιοϊατρικής. Για παράδειγμα, ιατρικές οντολογίες όπως η **Galen Medical Knowledge Base (GALEN)** [4] ή η **Gene Ontology (GO)** [5] είναι είτε ήδη καταγεγραμμένες σε $\mathcal{EL}$, είτε μπορούν να μετατραπούν εύκολα σε σώμα ορολογίας $\mathcal{EL}$. Η σημασία αυτής της οικογένειας περιγραφικών λογικών για τις σύγχρονες εφαρμογές καταδεικνύεται και από τη συμπερίληψη, ως ένα εκ των τριών διαθεσίμων **"προφίλ"** της **OWL 2 (OWL Web Ontology Language)**, δηλαδή διαθεσίμων συντακτικών και εκφραστικών υποσυνόλων, του "προφίλ" **OWL 2 EL** το οποίο βρίσκει λογικό υπόβαθρο στην οικογένεια των περιγραφικών λογικών $\mathcal{EL}$ ([6], [7]). Αυτό συνέβη λόγω της καταλληλότητας της οικογένειας για υπηρεσίες συλλογιστικής υψηλής απόδοσης σε πολυπληθή Σώματα Ορολογίας (TBoxes) ([8]). Μία εναλλακτική προσέγγιση στην προσπάθεια δημιουργίας μίας Περιγραφικής Λογικής με βατό αλγόριθμο συλλογιστικής παρουσιάζεται στο [9] και πρόκειται για την **DL-Lite**.

Η σύγχρονη αυτή τάση για δημιουργία μίας Περιγραφικής Λογικής με βατό αλγόριθμο συλλογιστικής έχει συνδυαστεί και με της οικογένεια των **Ασαφών Περιγραφικών Λογικών (Fuzzy Description Logics, f-DLs)**. Οι Ασαφείς Περιγραφικές Λογικές είναι φορμαλισμοί που έχουν προταθεί καθώς προσφέρουν τη δυνατότητας καταγραφής και συλλογισμών βάσει γνώσης ασαφούς ή στερούμενη ακρίβειας. Οι εφαρμογές στον **Σημασιολογικό Ιστό (Semantic Web)** ([10]) και οι ανάγκες για όσο το δυνατόν περισσότερο αυτόματη ανάκτηση, επεξεργασία, διανομή και επαναχρησιμοποίηση της πληροφορίας δείχνουν ότι σπάνια τα αποτελέσματα όλων αυτών των διαδικασιών μπορούν να αποτυπωθούν ως αληθή ή ψευδή. Αντίθετα, απαιτείται η χρήση βαθμών εμπιστοσύνης σε διαδικασίες όπως η συσχέτιση, η ομοιότητα και η ταξινόμηση. Το ίδιο ισχύει και για την επεξεργαστέα πληροφορία: οι έννοιες "ζεστό", "κρύο", "ακριβό", "κοντινό" αποτελούν παράδειγμα της παρουσιαζόμενης ασάφειας. Η υποστήριξη αυτού του είδους της ασαφούς πληροφορίας μπορεί να οδηγήσει σε πιο αποδοτικές, ρεαλιστικές και χρήσιμες εφαρμογές. ([11])

3.2 Σκοπός της εργασίας

Σκοπός της παρούσας εργασίας είναι η υλοποίηση μηχανής συλλογιστικής για την περιγραφική λογική $fuzzy - \mathcal{EL}^+$ ($f - \mathcal{EL}^+$) καθώς και τη κλασική (crisp) $\mathcal{EL}^+$, η οποία μπορεί να θεωρηθεί ειδική περίπτωση της ασαφούς. Ο αλγόριθμος συλλογιστικής για την $f - \mathcal{EL}^+$ παρουσιάζεται στο [12] και βασίζεται στην προσέγγιση που ακολουθείται στα [13] και [14] για την $\mathcal{EL}^+$. Ο αλγόριθμος που υλοποιήθηκε είναι ορθός, πλήρης και πολυωνυμικής πολυπλοκότητας.

Υλοποιήθηκε η βελτιστοποιημένη εκδοχή του αλγορίθμου, ενώ ιδιαίτερη προσοχή δόθηκε και σε περαιτέρω βελτιώσεις της υλοποίησης, ώστε οι τελικές επιδόσεις να είναι συγκρίσιμες με αντιστοίχων δυνατοτήτων συστήματα που παρουσιάζονται στη βιβλιογραφία, όπως η μηχανή συλλογιστικής CEL (Classifier for the description logic EL^+, [15]).

Ένας δεύτερος στόχος είναι η δυνατότητα ενσωμάτωσης και χρήσης της κατασκευασθείσας μηχανής συλλογιστικής από το σύστημα FReS (Fuzzy Reasoning Services). Το FReS είναι ένα σύστημα διαχείρισης ασαφούς πληροφορίας το οποίο υλοποιείται από την ερευνητική ομάδα των Τεχνολογιών Γνώσης στο Εργαστήριο Εικόνας, Βίντεο και Συστημάτων Πολυμέσων της Σχολής. Το σύστημα αυτό θα μπορεί να ανακτά και να αποθηκεύει πληροφορία σε βάσεις δεδομένων, θα παρέχει εργαλεία **ασαφοποίησης (fuzzification)** υπάρχουσας σαφούς πληροφορίας καθώς και εργαλεία για την **ευθυγράμμιση (alignment)** και το τη **διάσπαση σε υπομονάδες (modularisation)** οντολογιών, ενώ τέλος θα επιτρέπει μία πληθώρα υπηρεσιών ασαφούς συλλογιστικής για Περιγραφικές Λογικές όπως η $f-\mathcal{SHIN}$, η οικογένεια $\mathcal{EL}$ και η DL-Lite.

3.3 Συνεισφορά της Διπλωματικής Εργασίας

Στο πλαίσιο της παρούσας διπλωματικής εργασίας:

1. Μελετήθηκαν οι τρέχουσες εξελίξεις στον χώρο των Περιγραφικών Λογικών και των Τεχνολογιών Γνώσης με έμφαση στην οικογένεια των Περιγραφικών Λογικών $\mathcal{EL}$.
2. Έγινε ανασκόπηση του χώρου της ασαφούς συλλογιστικής, με έμφαση στην Περιγραφική Λογική $fuzzy-\mathcal{EL}^+$.
3. Υλοποιήθηκε ο αλγόριθμος συλλογιστικής για τις Περιγραφικές Λογικές $\mathcal{EL}^+$ και $fuzzy-\mathcal{EL}^+$, χρησιμοποιώντας τις παρουσιαζόμενες στη βιβλιογραφία βελτιστοποιήσεις καθώς και άλλες βελτιστοποιήσεις σε επίπεδο υλοποίησης.
4. Αξιολογήθηκε η απόδοση και η ορθότητα του υλοποιηθέντος συστήματος συλλογιστικής και τέλος,
5. Υλοποιήθηκαν οι κατάλληλες δομές επικοινωνίας με τα υλοποιούμενα στο Εργαστήριο άλλα συστήματα τεχνολογιών γνώσης.

3.4 Οργάνωση κειμένου

Στα επόμενα κεφάλαια:

1. θα παρουσιαστούν οι σχετικές δημοσιεύσεις και υλοποιήσεις στις οποίες βασίστηκε η εργασία,
2. θα περιγραφεί ο αλγόριθμος συλλογιστικής για σαφείς και ασαφείς οντολογίες,
3. θα γίνει περιγραφή της αρχιτεκτονικής και της σχεδίασης του συστήματος,
4. θα περιγραφεί συνοπτικά η υλοποίηση καθώς και ορισμένες τεχνικές επιτάχυνσης της εκτέλεσης του αλγορίθμου,
5. θα παρουσιαστούν παραδείγματα εκτέλεσης για τις διάφορες λειτουργίες της μηχανής συλλογιστικής και τέλος

6. θα παρουσιαστούν οι μετρήσεις του χρόνου που απαιτείται για την ταξινόμηση διαφόρων οντολογιών εισόδου, μαζί με συγκρίσεις για άλλες μηχανές συλλογιστικής. Τα αποτελέσματα θα σχολιασθούν και θα αναφερθούν προτάσεις για μελλοντικές επεκτάσεις και βελτιώσεις του προγράμματος.

Ο κώδικας της υλοποιηθείσας μηχανής συλλογιστικής όπως και η σχετική δομή σχολίων γι' αυτόν (Javadoc) μπορεί να βρεθεί στο συνοδευτικό CD στο οπισθόφυλλο του παρόντος αντιτύπου.

4. Ανάλυση Σχετικών Εργασιών

4.1 Η Περιγραφική Λογική $\mathcal{EL}^+$

Η μηχανή συλλογιστικής που κατασκευάστηκε υποστηρίζει την Περιγραφική Λογική $\mathcal{EL}^+$, οι έννοιες C της οποία σχηματίζονται σύμφωνα με τον ακόλουθο συντακτικό κανόνα:

$$C, D ::= A|\top|C \sqcap D|\exists r.C$$

όπου το A είναι όνομα έννοιας ($A \in CN$), το r όνομα ρόλου ($r \in RN$) και τα C, D έννοιες. Έτσι, όσον αφορά τις έννοιες, η γλώσσα $\mathcal{EL}^+$ είναι ισοδύναμη με την $\mathcal{EL}$. Η διαφορά τους έγκειται στο ότι μία οντολογία $\mathcal{EL}^+$ είναι ένα πεπερασμένο σύνολο Αξιωμάτων Υπαγωγής Γενικευμένων Εννοιών (General Concept Inclusions – GCIs) μορφής $C \sqsubseteq D$ και Αλυσίδων Ρόλων (Role Inclusions – RIs) μορφής $r_1 \circ ... \circ r_n \sqsubseteq r$. Αυτό σημαίνει ότι σε μία οντολογία $\mathcal{EL}^+$ μπορούν να εκφραστούν μεταβατικοί ρόλοι, ιεραρχίες ρόλων και οι ιδιότητες δεξιού μέλους (right-identities) σε ρόλους, δηλαδή αξιώματα της μορφής $r \circ s \sqsubseteq s$. Τα αξιώματα αυτού του τελευταίου τύπου είναι πολύ σημαντικά για την κατάρτιση βιοϊατρικών οντολογιών.

Για τη σημασιολογία της $\mathcal{EL}^+$ ισχύουν τα ακόλουθα:

Όνομα	**Σύνταξη**	**Σημασιολογία**
Top	$\top$	$\Delta^{\mathcal{I}}$
Σύζευξη	$C \sqcap D$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$
Υπαρξιακός Περιορισμός	$\exists r.C$	$\{x \in \Delta^{\mathcal{I}} \| y \in \Delta^{\mathcal{I}} : (x, y) \in r^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\}$
Αξίωμα Υπαγωγής Γενικευμένων Εννοιών (GCI)	$C \sqsubseteq D$	$C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$
Αλυσίδα Ρόλων (RI)	$r_1 \circ ... \circ r_n \sqsubseteq r$	$r_1^{\mathcal{I}} \circ ... \circ r_n^{\mathcal{I}} \subseteq r^{\mathcal{I}}$

Πίνακας 1: *Σημασιολογία EL*$^+$

Γράφοντας $C \sqsubseteq_{\mathcal{O}} D$ εννοούμε ότι η έννοια C υπάγεται στην έννοια D με βάση την οντολογία $\mathcal{O}$.

4.2 Η Περιγραφική Λογική f-$\mathcal{EL}^+$

Όπως έχει ήδη αναφερθεί, η μηχανή συλλογιστικής που κατασκευάστηκε υποστηρίζει και την Περιγραφική Λογική $f - \mathcal{EL}^+$ εκτός της κλασικής $\mathcal{EL}^+$. Η Περιγραφική Λογική $f - \mathcal{EL}^+$ παρουσιάζεται στο [12], ενώ η σημασιολογία και ο αλγόριθμος συλλογιστικής της

ασαφούς αυτής επέκτασης της $\mathcal{EL}^+$ κάνουν χρήση των τελεστών της λογικής Gödel. Για το λόγο αυτό η επέκταση που θα παρουσιαστεί καλείται $f_G - \mathcal{EL}^+$.

Ως συνήθως, οι περιγραφές εννοιών ορίζονται επαγωγικά από ένα σύνολο **ονομάτων εννοιών (concept names, CN)** και **ονομάτων ρόλων (role names, RN)**, σε συνδυασμό με ένα σύνολο κατασκευαστών. Συνεπώς, οι έννοιες C της $f_G - \mathcal{EL}^+$ σχηματίζονται σύμφωνα με τον ακόλουθο συντακτικό κανόνα:

$$C, D ::= A|\top|C \sqcap D|\exists r.C$$

όπου το A είναι όνομα έννοιας ($A \in CN$), το r όνομα ρόλου ($r \in RN$) και τα C, D έννοιες.

Μία οντολογία $f_G - \mathcal{EL}^+$αποτελείται από ένα πεπερασμένο σύνολο **αξιωμάτων**, δηλαδή κανόνων, **ρόλων** και **αξιωμάτων εννοιών**. Η διαφορά σε σύγκριση με τις σαφείς οντολογίες έγκειται στη δυνατότητα ύπαρξης ασαφών αξιωμάτων υπαγωγής γενικευμένων εννοιών (f-GCIs) τα οποία είναι της μορφής $\langle C \sqsubseteq D, n\rangle$ όπου $n \in (0, 1]$ και τα C, D είναι έννοιες της $f_G - \mathcal{EL}^+$. Ένα αξίωμα τέτοιας μορφής σημαίνει ότι ο βαθμός της υπαγωγής της έννοιας C στη D είναι ίσος με n. Από την άλλη πλευρά, δεν επιτρέπεται η χρήση ασαφών αξιωμάτων ρόλων: αυτά παραμένουν όπως και στην απλή $\mathcal{EL}^+$. Επομένως, οι αλυσίδες ρόλων (RIs) της $f_G - \mathcal{EL}^+$ είναι απλά αξιώματα της μορφής $r_1 \circ ... \circ r_k \sqsubseteq s$.

Η σημασιολογία των ασαφών περιγραφικών λογικών βασίζεται σε μία ασαφή ερμηνεία. Με τον όρο **Ασαφής Ερμηνεία (Fuzzy Interpretation)** νοείται ένα ζεύγος $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, όπου το πεδίο $\Delta^{\mathcal{I}}$ είναι ένα μη κενό σύνολο αντικειμένων και η $\cdot^{\mathcal{I}}$ είναι μία ασαφής συνάρτηση ερμηνείας, η οποία απεικονίζει:

1. ένα άτομο a σε ένα στοιχείο $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$,
2. ένα όνομα έννοιας A σε μία συνάρτηση συμμετοχής (membership function) $A^{\mathcal{I}} : \Delta^{\mathcal{I}} \to [0, 1]$ και
3. ένα όνομα ρόλου r σε μία συνάρτηση συμμετοχής $r^{\mathcal{I}} : \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \to [0, 1]$.

Χρησιμοποιώντας τους τελεστές (operators) των ασαφών συνόλων, οι ασαφείς ερμηνείες μπορούν να επεκταθούν ώστε να ερμηνεύουν και τις έννοιες της $f_G - \mathcal{EL}^+$. Το πλήρες σύνολο της σημασιολογίας της Περιγραφικής Λογικής παρατίθεται στον **Πίνακα 2**, όπου ως $\mathcal{J}_G$ συμβολίζουμε την **ασαφή συνεπαγωγή Gödel (Gödel fuzzy implication)**. Η επιλογή αυτή είναι κατάλληλη, αφού $\langle C \sqsubseteq D, 1\rangle$ αν και μόνο αν $C^{\mathcal{I}}(a) \leq D^{\mathcal{I}}(a), \forall a \in \Delta^{\mathcal{I}}$. Αυτό διαισθητικά σημαίνει ότι αν η έννοια C υπάγεται πλήρως από την D, τότε η συνάρτηση συμμετοχής της D είναι μεγαλύτερη ή ίση αυτής της C σε όλες τις περιπτώσεις. Επιπλέον, αυτό έχει ως συνέπεια την πλήρη υπαγωγή κάθε έννοιας C από τον εαυτό της.

Όνομα	Σύνταξη	Σημασιολογία
Top	$\top$	$\top^{\mathcal{I}}(a) = 1$
Σύζευξη	$C \sqcap D$	$(C \sqcap D)^{\mathcal{I}}(a) = min(C^{\mathcal{I}}(a), D^{\mathcal{I}}(a))$
Υπαρξιακός Περιορισμός	$\exists r.C$	$(\exists r.C)^{\mathcal{I}}(a) = \sup_{b \in \Delta^{\mathcal{I}}}\{min(r^{\mathcal{I}}(a,b), C^{\mathcal{I}}(b))\}$
Αξίωμα Υπαγωγής Γενικευμένων Εννοιών (GCI)	$\langle C \sqsubseteq D, n\rangle$	$\inf_{a \in \Delta^{\mathcal{I}}} \mathcal{J}_G(C^{\mathcal{I}}(a), D^{\mathcal{I}}(a)) \geq n$
Αλυσίδα Ρόλων (RI)	$r_1 \circ ... \circ r_k \sqsubseteq s$	$[r_1^{\mathcal{I}} \circ^t ... \circ^t r_k^{\mathcal{I}}](a,b) \leq s^{\mathcal{I}}(a,b)$

Πίνακας 2: *Σημασιολογία f_G-EL$^+$*

Το κύριο πρόβλημα συλλογιστικής για την $f_G - \mathcal{EL}^+$είναι η ασαφής υπαγωγή εννοιών: υποστηρίζουμε ότι μία έννοια C υπάγεται ασαφώς από την D κατά ένα βαθμό $n \in (0, 1]$ με βάση μία $f_G - \mathcal{EL}^+$ οντολογία $\mathcal{O}$ και το συμβολίζουμε με $\langle C \sqsubseteq_{\mathcal{O}} D, n\rangle$, αν ισχύει η σχέση $\inf_{a \in \Delta^{\mathcal{I}}} \mathcal{J}_G(C^{\mathcal{I}}(a), D^{\mathcal{I}}(a)) \geq n$ για κάθε μοντέλο $\mathcal{I}$ της $\mathcal{O}$. Επιπλέον, μας ενδιαφέρει το πρόβλημα της **ταξινόμησης (classification)** μίας $f_G - \mathcal{EL}^+$ οντολογίας, δηλαδή ο υπολογισμός όλων των ασαφών σχέσεων υπαγωγής μεταξύ των εννοιών της οντολογίας.

Η περιγραφική λογική $f_G - \mathcal{EL}^+$ θα αναφέρεται στο εξής απλά ως $f - \mathcal{EL}^+$.

4.3 OWL 2 EL

Το βασικό γεγονός που αναδεικνύει τη χρησιμότητα της οικογένειας Περιγραφικών Λογικών $\mathcal{EL}$ για τις σύγχρονες εφαρμογές στο χώρο των Τεχνολογιών Γνώσης είναι η συμπερίληψη της οικογένειας σε ένα εκ των τριών διαθεσίμων **"προφίλ"** της **OWL 2 (OWL Web Ontology Language)**. Με τον όρο "προφίλ" νοείται ένα συντακτικό και εκφραστικό υποσύνολο της OWL 2. Το "προφίλ" OWL 2 EL βρίσκει λογικό υπόβαθρο στην οικογένεια των περιγραφικών λογικών $\mathcal{EL}$ και συγκεκριμένα στην $\mathcal{EL}^{++}$ ([6], [7]). Έτσι, η ύπαρξη Σώματος Ορολογίας (Terminological Box, Tbox) και Σώματος Ισχυρισμών (Assertional Box – Abox) στην $\mathcal{EL}^{++}$ συνεπάγεται την ύπαρξη συνόλου αξιωμάτων κλάσεων, ιδιοτήτων αντικειμένων (object properties) και ατόμων (individuals) στην OWL 2 EL.

Όπως έχει ήδη αναφερθεί, η επιλογή της οικογένειας $\mathcal{EL}$ έγινε εξ' αιτίας της ιδιότητάς της να παρέχει πολύ ισχυρή εκφραστικότητα – ιδιαιτέρως λόγω των κανόνων αλυσίδων ρόλων –, διαθέτοντας ταυτόχρονα αλγόριθμο συλλογιστικής πολυωνυμικής πολυπλοκότητας ως προς το μέγεθος της οντολογίας. Όπως και στην $\mathcal{EL}^+$, έτσι και στην OWL 2 EL η βασική υπηρεσία συλλογιστικής είναι η ταξινόμηση (classification) των κλάσεων οντολογίας, δηλαδή ο υπολογισμός όλων των σχέσεων υπαγωγής μεταξύ αυτών.

Προκειμένου να είναι δυνατή η καταγραφή μίας $\mathcal{EL}$ οντολογίας σε OWL 2 EL και, συνεπώς, σε γλώσσα OWL 2, γίνονται οι ακόλουθες συμβάσεις:

1. Χρησιμοποιείται ο τελεστής της ένωσης.

2. Ως υπαρξιακός ποσοδείκτης χρησιμοποιείται ο περιορισμός SomeValuesFrom της OWL 2.
3. Προκειμένου να διατηρηθεί η πολυωνυμική πολυπλοκότητα της συλλογιστικής, δεν επιτρέπεται η χρήση άρνησης, τομής, καθολικών ποσοδεικτών και περιορισμών πληθικότητας.

Εκ πρώτης όψεως οι περιορισμοί αυτοί δείχνουν να περιορίζουν σημαντικά την εκφραστικότητα, στην πραγματικότητα όμως υπάρχουν πολυάριθμες σημαντικές και εκτεταμένες οντολογίες οι οποίες μπορούν να καταγραφούν στη μορφή αυτή. Ένα παράδειγμα είναι η βιοϊατρική οντολογία SNOMED CT (Systematized Nomenclature of Medicine – Clinical Terms). Η ισχυρή αυτή εκφραστική δυνατότητα στην πράξη βασίζεται στη δυνατότητα ορισμού εννοιών με χρήση ενός συγκεκριμένου μοτίβου συνδυασμένης χρήσης τελεστών ένωσης και υπαρξιακών ποσοδεικτών. Για παράδειγμα, είναι πολύ εύκολο να εκφράσουμε στην οντολογία τη γνώση ότι μία καρδιά διαθέτει ως τμήματά της την αριστερή και τη δεξιά κοιλία.

4.4 OWL API

Το OWL API είναι μία διαπροσωπεία (interface) και υλοποίηση της Web Ontology Language (OWL) του W3C (World Wide Web Consortium) στη γλώσσα προγραμματισμού Java ([17], [18]), η οποία υλοποιήθηκε κατά κύριο λόγο από το Πανεπιστήμιο του Manchester. Επιτρέπει τον χειρισμό οντολογιών γραμμένων σε OWL 2 και είναι ελεύθερο λογισμικό που διατίθεται με τους όρους της GNU LGPL (GNU Lesser General Public License).

Το OWL API παρέχει δυνατότητες ανάγνωσης, επεξεργασίας και αποθήκευσης οντολογιών σε διάφορες μορφές, όπως OWL/XML, RDF/XML, OWL Functional Syntax και άλλες ([19]). Παρέχει μεθόδους και δομές δεδομένων για τη δημιουργία, χειρισμό, επεξεργασία και αποθήκευση οντοτήτων που εμφανίζονται σε μία οντολογία, όπως έννοιες, ρόλοι, αξιώματα, περιγραφές, επισημειώσεις (annotations) και πολλές άλλες. Επιτρέπει επίσης την εκτέλεση διαφόρων απλών εργασιών σε μία οντολογία, εκ των οποίων ορισμένες από τις πιο χρήσιμες για την υλοποίηση της μηχανής συλλογιστικής για την $f - \mathcal{EL}^+$ είναι:

- η δυνατότητα ελέγχου της εκφραστικότητας μίας οντολογίας,
- η εμφάνιση αξιωμάτων (όλων ή μόνο συγκεκριμένου τύπου – για παράδειγμα μόνο αξιώματα υπαγωγής κλάσεων ή μόνο ισοδυναμίας ρόλων),
- η εξαγωγή τμημάτων από κάποιο αξίωμα ή περιγραφή, όπως για παράδειγμα εξαγωγή του ρόλου από έναν υπαρξιακό περιορισμό ή της υπερκλάσης από ένα αξίωμα υπαγωγής,
- η δημιουργία νέων αξιωμάτων, περιγραφών και οντολογιών και
- η δυνατότητα εύρεσης υποκλάσεων / υπερκλάσεων και υπορόλων / υπερρόλων βάσει των αξιωμάτων της οντολογίας και όχι από παραγόμενη γνώση.

Μεταξύ ενός όρου των Περιγραφικών Λογικών και των αντικειμένων – κλάσεων του OWL API, ισχύουν αντιστοιχίσεις όπως αυτές που παρουσιάζονται στον παρακάτω **Πίνακα 3**. Αναφέρονται μόνο εκείνοι οι όροι που είναι χρήσιμοι για την $f - \mathcal{EL}^+$, ενώ όλες οι αναφερόμενες κλάσεις ανήκουν στο πακέτο org.semanticweb.owl.model.

Όρος Περιγραφικών Λογικών	**Σύνταξη**	**Κλάση**
Οτιδήποτε	-	OWLObject
Έννοια	C	OWLClass
Ρόλος	r	OWLObjectPropertyExpression
Υπαρξιακός περιορισμός	$\exists r.B$	OWLObjectSomeRestriction
Περιγραφή	π.χ. $C \sqcap \exists r.D$	OWLDescription
Αξίωμα ή Κανόνας	-	OWLAxiom
Αξίωμα Υπαγωγής Εννοιών	$C \sqsubseteq D$	OWLSubClassAxiom
Αξίωμα Ισοδυναμίας Εννοιών	$C \equiv D$	OWLEquivalentClassesAxiom
Αξίωμα Υπαγωγής Ρόλων	$r \sqsubseteq s$	OWLObjectSubPropertyAxiom
Αλυσίδα Ρόλων	$r_1 \circ ... \circ r_n \sqsubseteq r$	OWLObjectPropertyChainSub PropertyAxiom

Πίνακας 3: *Αντιστοιχίσεις όρων EL$^+$ με κλάσεις του OWL API*

4.5 Συστήματα Συλλογιστικής

4.5.1 CEL

Ένα σύστημα συλλογιστικής για την οικογένεια Περιγραφικών Λογικών $\mathcal{EL}$ είναι ο CEL, ο οποίος αναπτύχθηκε στο Technische Universität Dresden της Γερμανίας ([15]). Αρχικά ήταν σε θέση να διαχειριστεί οντολογίες σε Περιγραφική Λογική $\mathcal{EL}^+$, ενώ στις τελευταίες εκδόσεις υποστηρίζει και την $\mathcal{EL}^{++}$, η οποία διαθέτει την έννοια bottom (bottom concept, $\bot$) – οπότε μπορούν να ορισθούν και ξένες έννοιες –, restricted concrete domains και άτομα – συνεπώς και Σώμα Ισχυρισμών (Assertional Box - ABox). Ο αλγόριθμος συλλογιστικής είναι πολυωνυμικής χρονικής πολυπλοκότητας και παρουσιάζεται στο [8]. Είναι ο ίδιος που ακολουθείται για την Περιγραφική Λογική $\mathcal{EL}^+$ από τη μηχανή συλλογιστικής που υλοποιήθηκε και θα παρουσιαστεί λεπτομερώς και στο επόμενο κεφάλαιο, "Σφάλμα: Δεν βρέθηκε η πηγή παραπομπής". Ουσιαστικά, ο αλγόριθμος αυτός επιτρέπει την **ταξινόμηση (classification)** των εννοιών της οντολογίας.

Η πορεία που ακολουθεί ο CEL για να ταξινομήσει μία οντολογία είναι η ίδια με αυτή που ακολουθεί η υλοποιηθείσα μηχανή συλλογιστικής:

1. Αρχικά γίνεται εισαγωγή της οντολογίας. Όταν υλοποιήθηκε για πρώτη φορά ο CEL, η τότε τελευταία έκδοση της OWL δεν υποστήριζε πλήρως τα εκφραστικά μέσα της $\mathcal{EL}^+$. Συγκεκριμένα, δεν ήταν δυνατό να εκφραστούν αξιώματα της μορφής $r \circ s \sqsubseteq t$ (Αλυσίδες Ρόλων, Role Inclusions). Για τον λόγο αυτό αποφασίστηκε η χρήση μίας μικρής επέκτασης της KRSS αναπαράστασης οντολογιών ως χρησιμοποιούμενης μορφής εισόδου. Με τη δεύτερη έκδοση της OWL αυτή η έλλειψη εκφραστικότητας έπαψε να υπάρχει, παρ' όλα αυτά ο CEL ακόμα υποστηρίζει τη συλλογιστική μόνο σε οντολογίες

καταγεγραμμένες με την ειδική για αυτόν μορφή αναπαράστασης. Σχεδιάζεται όμως η υλοποίηση ενός αυτόματου μετατροπέα από τη σύνταξη της OWL στη σύνταξη του CEL. ([7])

2. Γίνεται μετατροπή της οντολογίας στην κανονική της μορφή, όπως αυτή παρουσιάζεται στην ενότητα "Μετατροπή σε Κανονική Μορφή Οντολογιών EL+" (σελ. 25). Η μετατροπή αυτή γίνεται σε χρόνο γραμμικό ως προς το μέγεθος των αξιωμάτων εισόδου, ενώ το μέγεθος της παραγόμενης οντολογίας είναι και αυτό γραμμικό ως προς το μέγεθος της αρχικής ([14]). Στόχος της μετατροπής αυτής είναι η διατήρηση όλων των σχέσεων υπαγωγών της αρχικής οντολογίας.
3. Στο επόμενο στάδιο υπολογίζονται τα σύνολα απεικονίσεων (mappings) S και R, όπως ακριβώς και στον υλοποιηθέντα αλγόριθμο. Σκοπός του σταδίου αυτού είναι η καταγραφή όλων των υπονοούμενων σχέσεων υπαγωγής μεταξύ εννοιών της οντολογίας. Ο αλγόριθμος αυτού του σταδίου είναι πολυωνυμικός ($O(n^4)$).
4. Σε επόμενες εκδόσεις του CEL προστέθηκε και αλγόριθμος υπολογισμού των άμεσων σχέσεων υπαγωγής μεταξύ των εννοιών της οντολογίας. Το στάδιο αυτό είναι όμοιο με το αντίστοιχο στάδιο του συστήματος που υλοποιήθηκε για την παρούσα διπλωματική εργασία.

Ο κώδικας του CEL βρίσκεται ελεύθερα διαθέσιμος στο [20]. Ο CEL έχει υλοποιηθεί σε Common LISP και επιτυγχάνει αρκετά καλές επιδόσεις σε σύγκριση με άλλες μηχανές συλλογιστικής, όπως ο FaCT++, ο Racer και ο Pellet ([15]). Στο σημείο αυτό θα πρέπει βέβαια να αναφέρουμε ότι οι υπόλοιπες μηχανές συλλογιστικής υποστηρίζουν Περιγραφικές Λογικές που αποτελούν εν γένει εκφραστικά υπερσύνολα της οικογένειας $\mathcal{EL}$, με αποτέλεσμα οι αλγόριθμοι που ακολουθούν να μην είναι τόσο βελτιστοποιημένοι όσο ο αλγόριθμος που υλοποιείται σε εξειδικευμένες μηχανές συλλογιστικής. Πιο συγκεκριμένα, οι περισσότερες σύγχρονες μηχανές συλλογιστικής χρησιμοποιούν αλγορίθμους tableau, οι οποίοι στη χειρότερη περίπτωση εμφανίζουν τουλάχιστον εκθετική χρονική πολυπλοκότητα. Παρά την ανάπτυξη όμως πολυάριθμων τεχνικών βελτιστοποίησης, η απόδοση αυτών δε μπορεί να συγκριθεί ακόμα και με απλές υλοποιήσεις του αλγορίθμου που χρησιμοποιείται στον CEL. Βέβαια, ορισμένες από τις μηχανές συλλογιστικής, όπως ο Pellet, χρησιμοποιούν ειδικούς αλγορίθμους χειρισμού οντολογιών αναλόγως της εκφραστικότητάς τους, ώστε να επιλέγεται πάντα ο προτιμότερος αλγόριθμος για το είδος της οντολογίας και της υπηρεσίας συλλογιστικής που απαιτείται.

Ο CEL, όταν υλοποιήθηκε η πρώτη έκδοσή του το 2005 ([14]), ήταν το μόνο σύστημα Περιγραφικών Λογικών υλοποιηθέν στον ακαδημαϊκό χώρο το οποίο μπορούσε να ταξινομήσει την ιατρική οντολογία **SNOMED-CT (Systematized Nomenclature of Medicine – Clinical Terms)** ([16]). Το γεγονός αυτό από τη μία πλευρά κινητοποίησε τους ερευνητές στον χώρο των Περιγραφικών Λογικών ώστε να ανακαλύψουν τεχνικές βελτιστοποίησης ειδικά σχεδιασμένες για βιοϊατρικές οντολογίες, και ιδίως για τη SNOMED CT, ενώ από την άλλη πλευρά οδήγησε μηχανές συλλογιστικής βασισμένες σε αλγορίθμους tableau, όπως ο FaCT++ και ο RacerPro, να υιοθετήσουν ορισμένες από αυτές τις τεχνικές και να βελτιώσουν την απόδοσή τους.

Σε μετέπειτα υλοποιήσεις η αποδοτικότητα του CEL βελτιώθηκε περαιτέρω, ενώ εισήχθησαν και επιπλέον δυνατότητες όπως η πλήρης υποστήριξη της $\mathcal{EL}^{++}$, η δυνατότητα αυξητικής ταξινόμησης (incremental classification), η δυνατότητα διάσπασης της οντολογίας σε υπομονάδες (modularisation) και η υποστήριξη συλλογιστικής σε πολύπλοκες έννοιες – δηλαδή κάνοντας χρήση ερωτημάτων πολυπλοκότερων από τον απλό έλεγχο υπαγωγής ονομάτων

κλάσεων ([21]). Η τελευταία αυτή δυνατότητα υποστηρίζεται και από την υλοποιηθείσα στα πλαίσια της παρούσας διπλωματικής εργασίας μηχανή συλλογιστικής.

Ο CEL είναι δυνατόν να ταξινομήσει μία οντολογία και να εξάγει τις σχετικές πληροφορίες με την παρακάτω εντολή:

```
cel -loadOntology [file]  -classifyOntology -outputTaxonomy [file]
```

Το πρώτο όρισμα καθορίζει το αρχείο όπου βρίσκεται η οντολογία εισόδου, το δεύτερο δίνει εντολή ταξινόμησης της οντολογίας και το τρίτο εξάγει τις σχετικές πληροφορίες (άμεσες υπερκλάσεις και υποκλάσεις) σε κάποιο αρχείο.

4.5.2 FaCT++

Από τα υπόλοιπα συστήματα συλλογιστικής τις καλύτερες επιδόσεις επιτυγχάνει ο FaCT++, ο οποίος έχει υλοποιηθεί στη γλώσσα προγραμματισμού C++ και βασίζεται σε αλγορίθμους tableau. Υποστηρίζει την περιγραφική λογική $\mathcal{SROIQ}$ (OWL 1.1) ([22]). Η βασική διαφορά του FaCT++ σε σύγκριση με άλλες μηχανές συλλογιστικής έγκειται στην ενσωμάτωση βελτιστοποιήσεων τόσο στην προεπεξεργασία, στην κανονικοποίηση και στην αποδοτική αναπαράσταση της οντολογίας, όσο και στη διαδικασία της συλλογιστικής. Προσφέρονται υπηρεσίες ελέγχου ικανοποιησιμότητας και ελέγχου υπαγωγής εννοιών. Η σημαντική αύξηση των επιδόσεων της μηχανής οφείλεται κυρίως στην ενσωμάτωση ευρετικών (heuristic) τεχνικών για την επιλογή του επόμενου κανόνα και της έννοιας στην οποία αυτός θα εφαρμοστεί. ([23])

Και οι δύο μηχανές συλλογιστικής (CEL, FaCT++) υποστηρίζουν τη διαπροσωπεία DIG (DL Implementation Group). Η διαπροσωπεία αυτή προσφέρει έναν κοινό τρόπο πρόσβασης και επικοινωνίας με μηχανές συλλογιστικής για Περιγραφικές Λογικές. Συγκεκριμένα, ορίζει ένα απλό πρωτόκολλο επικοινωνίας βασιζόμενο στις μεθόδους PUT/GET του HTTP μαζί με ένα σχήμα XML που περιγράφει μία νοητή γλώσσα και ορισμένες λειτουργίες. Η διαπροσωπεία DIG δε στοχεύει στην υποστήριξη περίπλοκων υπηρεσιών συλλογιστικής, αλλά αντιθέτως προσφέρει μόνο ορισμένες απλές υπηρεσίες, όπως έλεγχο ικανοποιησιμότητας, υπαγωγής και ταξινόμησης (classification). Οι υπηρεσίες αυτές είναι οι συχνότερα χρησιμοποιούμενες στην πράξη. Η διαπροσωπεία DIG υποστηρίζεται από ένα σύνολο μηχανών συλλογιστικής, όπως οι CEL, FaCT++, Pellet και RacerPro, ενώ χρησιμοποιείται και από ένα σύνολο επεξεργαστών οντολογιών όπως οι OilEd, Protégé και SWOOP, επιτρέποντας τη μεταξύ αυτών επικοινωνία ([24]).

4.6 Whirly Cache

Κατά την υλοποίηση και το μετέπειτα έλεγχο του αλγορίθμου για τα σημεία εκείνα που εισάγουν τη μεγαλύτερη καθυστέρηση κατά την εκτέλεση (bottlenecks) παρατηρήθηκε πως υπήρχαν μέθοδοι που καλούνταν πολλές φορές κατά τη διάρκεια της εκτέλεσης του προγράμματος με τα ίδια ορίσματα, ενώ το αποτέλεσμα εξαρτάτο μόνο από αυτά τα αρχικά δεδομένα: ίδια είσοδος σήμαινε ίδιο αποτέλεσμα. Επειδή ο επανυπολογισμός κάθε φορά του αποτελέσματος αποτελούσε ένα πολύ σημαντικό τμήμα του χρόνου εκτέλεσης του προγράμματος, έγινε αρχικά προσπάθεια αποθήκευσης κάθε ζεύγους εισόδου αποτελέσματος στη μνήμη, την πρώτη φορά

που αυτό θα υπολογιζόταν. Μία τέτοια λύση ήταν ικανοποιητική για μικρές οντολογίες, όμως σε μεγαλύτερες εμφανίζονταν προβλήματα χρήσης μνήμης. Για το λόγο αυτό, αντί της πλήρους αποθήκευσης κάθε αποτελέσματος υιοθετήθηκε η χρήση μίας ενδιάμεσης **κρυφής μνήμης (cache memory)**. Το μειονέκτημα της μεγάλης απαίτησης σε μνήμη μειώθηκε σημαντικά, καθώς είναι δυνατό να οριστεί το επιθυμητό μέγιστο πλήθος εγγραφών στην κρυφή μνήμη, ενώ η ταχύτητα του προγράμματος παρέμεινε πρακτικά η ίδια με την υλοποίηση της πλήρους αποθήκευσης.

Αντί μίας νέας υλοποίησης οργάνωσης κρυφής μνήμης χρησιμοποιήθηκε η έτοιμη βιβλιοθήκη Whirlycache, η οποία αποτελεί μία γρήγορη κρυφή μνήμη (cache memory) για αντικείμενα οποιουδήποτε τύπου γραμμένη σε γλώσσα Java ([25]). Χρησιμοποιείται για να αποθηκεύει αντικείμενα τα οποία ζητούνται συχνά και η επαναδημιουργία τους είναι μία χρονοβόρος διαδικασία. Παρουσιάζει πολύ καλές επιδόσεις σε σύγκριση με άλλες υλοποιήσεις κρυφών μνημών σε Java, ιδίως λόγω ορισμένων σχεδιαστικών χαρακτηριστικών της.

Ο βασικός σχεδιαστικός στόχος της Whirlycache είναι ότι οι διαδικασίες εισαγωγής (insertion) και ανάκτησης (retrieval) αντικειμένων στη μνήμη θα πρέπει να εκτελούνται όσο το δυνατόν ταχύτερα, χωρίς να επιβαρύνουν με καθυστερήσεις το υπόλοιπο πρόγραμμα. Επίσης, πρέπει να τονιστεί πως η Whirlycache σχεδιάστηκε ώστε να είναι κατάλληλη για εφαρμογές οι οποίες δεν απαιτούν εγγυημένη εκτέλεση μίας λειτουργίας εντός συγκεκριμένου χρονικού πλαισίου (hard limit), αλλά αρκούνται σε μια "καλύτερη προσπάθεια" (best effort – soft limit). Τέλος, βασικό ρόλο για τη λειτουργία και τις επιδόσεις της κρυφής μνήμης έχει ένα νήμα (thread) το οποίο εκτελείται στο παρασκήνιο (background): το νήμα αυτό ονομάζεται Tuner και εκτελείται περιοδικά – με περίοδο οριζόμενη από τον προγραμματιστή –, εφαρμόζοντας στη μνήμη τις αλλαγές που έπρεπε να γίνουν εξ' αιτίας των κλήσεων που έλαβαν χώρα μεταξύ δύο εκτελέσεων. Ουσιαστικά, ο Tuner διαγράφει αντικείμενα από την μνήμη βάσει του ορισθέντος επιθυμητού μεγέθους της και της ορισθείσας πολιτικής διαγραφών. Αυτό έχει ως συνέπεια ναι μεν να είναι οι λειτουργίες εισαγωγής και ανάκτησης ταχείες, αλλά να υπάρχει και το ενδεχόμενο μεταξύ δύο εκτελέσεων του Tuner νήματος το πλήθος των καταχωρίσεων στη μνήμη να αυξηθεί πέραν του ορισθέντος μεγίστου. Κάτι τέτοιο βέβαια δεν είναι απαραίτητα πρόβλημα, ιδίως αν το νήμα εκτελείται αρκετά συχνά για τα δεδομένα της εφαρμογής.

Η Whirlycache υποστηρίζει τρεις διαφορετικές πολιτικές αντικατάστασης στοιχείων: FIFO (First In First Out), LRU (Least Recently Used) και LFU (Least Frequently Used).

5. Συλλογιστική στην $\mathcal{EL}^+$

Όπως έχει ήδη αναφερθεί προηγουμένως, για την παρούσα διπλωματική εργασία υλοποιήθηκε αλγόριθμος συλλογιστικής για την περιγραφική λογική $\mathcal{EL}^+$ και για την ασαφή επέκτασή της $f - \mathcal{EL}^+$.

Ο αλγόριθμος συλλογιστικής για την $\mathcal{EL}^+$ που υλοποιήθηκε στην παρούσα μηχανή συλλογιστικής παρουσιάζεται στα [13] και [14], ενώ βασίζεται στον αλγόριθμο για την $\mathcal{EL}^{++}$ που παρουσιάστηκε στο [8]. Ανήκει στην οικογένεια των αλγορίθμων δομικής υπαγωγής και στηρίζεται στους αλγορίθμους που έχουν προταθεί για την οικογένεια περιγραφικών λογικών $\mathcal{FL}$. Το βασικό πρόβλημα συλλογιστικής για περιγραφές εννοιών είναι η υπαγωγή εννοιών: μία έννοια C υπάγεται σε μία έννοια D με βάση μία οντολογία $\mathcal{O}$ (δηλαδή $C \sqsubseteq_{\mathcal{O}} D$) αν ισχύει $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ για κάθε μοντέλο $\mathcal{I}$ της $\mathcal{O}$. Συνεπώς, το βασικό πρόβλημα συλλογιστικής σε επίπεδο οντολογίας είναι η ταξινόμηση (classification), δηλαδή ο υπολογισμός της ιεραρχίας υπαγωγών για όλες τις ονοματικές έννοιες που εμφανίζονται σε αυτήν.

Από την άλλη πλευρά, ο αλγόριθμος συλλογιστικής για την $f - \mathcal{EL}^+$ όπως παρουσιάζεται στο [12] δεν είναι σημαντικά διαφορετικός σε σύγκριση με τον κλασικό αλγόριθμο. Η διαδικασία που ακολουθείται για την ταξινόμηση της οντολογίας είναι αντίστοιχη, ενώ αξίζει να αναφερθεί πως ο σαφούς λογικής αλγόριθμος θα μπορούσε να ιδωθεί ως ειδική περίπτωση του αλγορίθμου ασαφούς λογικής: απλά όλοι οι βαθμοί των αξιωμάτων υπαγωγής εννοιών θα ήταν ίσοι με τη μονάδα. Χάριν πληρότητας όμως παρουσιάζονται και οι δύο αλγόριθμοι ανεξάρτητα στο πλαίσιο του παρόντος κεφαλαίου.

Στο κεφάλαιο αυτό δε θα παρουσιαστούν λεπτομέρειες της υλοποίησης: θα γίνει περιγραφή μόνο σε επίπεδο αλγορίθμου.

5.1 Ταξινόμηση Οντολογίας $\mathcal{EL}^+$

Βάσει των [13] και [14], πρώτα θα παρουσιαστεί ένας αφηρημένος αλγόριθμος ταξινόμησης (classification) πολυωνυμικού χρόνου για την περιγραφική λογική $\mathcal{EL}^+$, ενώ στη συνέχεια θα παρουσιαστεί μία πιο συγκεκριμένη έκδοση του ιδίου αλγορίθμου, η οποία επιτυγχάνει μειωμένη θεωρητική χρονική πολυπλοκότητα – πάλι όμως πολυωνυμική – και συνεπώς υψηλότερη απόδοση.

Τόσο σε συστήματα συλλογιστικής Περιγραφικών Λογικών βασιζόμενα σε αλγορίθμους tableau, όσο και στα αρχικά συστήματα βασιζόμενα σε αλγορίθμους δομικής υπαγωγής, η ιεραρχία υπαγωγών υπολογίζεται εκτελώντας πολλαπλούς ελέγχους υπαγωγής εννοιών. Αυτό σημαίνει ότι σε τέτοια συστήματα υπάρχει ανάγκη όχι μόνο βελτιστοποίησης της διαδικασίας ελέγχου της ισχύος υπαγωγής ενός ζεύγους εννοιών, αλλά και περιορισμού του πλήθους των ελέγχων υπαγωγής ζευγών που απαιτούνται προκειμένου να υπολογισθεί η συνολική ιεραρχία. Αντιθέτως, ο αλγόριθμος υπαγωγής που παρουσιάστηκε στο [8] υπολογίζει ταυτόχρονα τις σχέσεις υπαγωγής μεταξύ όλων των ζευγών ονομάτων εννοιών της οντολογίας εισόδου.

5.1.1 Μετατροπή σε Κανονική Μορφή Οντολογιών $\mathcal{EL}^+$

Προτού περιγραφεί τον πολυωνυμικού χρόνου αλγόριθμο ταξινόμησης για την $\mathcal{EL}^+$ πρέπει να ορισθεί μία κατάλληλη κανονική μορφή για την οντολογία.

Γενικά, σε μία οντολογία $\mathcal{O}$, ορίζουμε με τους συμβολισμούς $CN_{\mathcal{O}}^{\top}$ και $CN_{\mathcal{O}}$ τα σύνολα των ονομάτων εννοιών που εμφανίζονται στην $\mathcal{O}$, συμπεριλαμβανομένης ή μη συμπεριλαμβανομένης της έννοιας top ($\top$) αντίστοιχα.

Λέμε λοιπόν ότι η $\mathcal{O}$ είναι σε κανονική μορφή εάν:

1. Όλα τα γενικευμένα αξιώματα υπαγωγής εννοιών (GCIs) της οντολογίας λαμβάνουν μία εκ των ακολούθων μορφών, όπου $A_i \in CN_{\mathcal{O}}^{\top}$ και $B \in CN_{\mathcal{O}}$:

$$A_1 \sqcap ... \sqcap A_n \sqsubseteq B,$$

$$A_1 \sqsubseteq \exists r.A_2 \text{ και}$$

$$\exists r.A_1 \sqsubseteq B.$$

2. Όλα τα αξιώματα υπαγωγής ρόλων (RIs) είναι της μορφής:

$$r \sqsubseteq s \text{ ή } r_1 \circ r_2 \sqsubseteq s.$$

Εισάγοντας νέα ονόματα ρόλων και εννοιών, κάθε $\mathcal{EL}^+$ οντολογία $\mathcal{O}$ μπορεί να μετασχηματιστεί σε μία **κανονικοποιημένη οντολογία** $\mathcal{O}'$ η οποία να είναι **συντηρητική επέκταση** της $\mathcal{O}$. Με τον όρο συντηρητική επέκταση εννοούμε πως αν $C \sqsubseteq_{\mathcal{O}} D$ τότε και μόνο τότε $C \sqsubseteq_{\mathcal{O}'} D$ για όλες τις περιγραφές εννοιών C, D που μπορούν να κατασκευαστούν από τα ονόματα εννοιών και ρόλων που συναντώνται στην $\mathcal{O}$. Μάλιστα, αποδεικνύεται ότι **ο μετασχηματισμός αυτός μπορεί να λάβει χώρα σε γραμμικό χρόνο** ως προς το μέγεθος της οντολογίας. Μία λεπτομερής απόδειξη είναι δυνατό να βρεθεί στο [26].

Κανόνας Κανονικοποίησης	**Αντικατάσταση**
NF1	$r_1 \circ ... \circ r_k \sqsubseteq s \rightsquigarrow r_1 \circ ... \circ r_{k-1} \sqsubseteq u, u \circ r_k \sqsubseteq s$
NF2	$C_1 \sqcap ... \sqcap \hat{C} \sqcap ... \sqcap C_n \sqsubseteq D \rightsquigarrow \hat{C} \sqsubseteq A, C_1 \sqcap ... \sqcap A \sqcap ... \sqcap C_n \sqsubseteq D$
NF3	$\exists r.\hat{C} \sqsubseteq D \rightsquigarrow \hat{C} \sqsubseteq A, \exists r.A \sqsubseteq D$
NF4	$\hat{C} \sqsubseteq \hat{D} \rightsquigarrow \hat{C} \sqsubseteq A, A \sqsubseteq \hat{D}$
NF5	$B \sqsubseteq \exists r.\hat{C} \rightsquigarrow B \sqsubseteq \exists r.A, A \sqsubseteq \hat{C}$
NF6	$B \sqsubseteq C \sqcap D \rightsquigarrow B \sqsubseteq C, B \sqsubseteq D$

***Πίνακας 4**: Κανόνες κανονικοποίησης οντολογιών EL+*

Η κανονικοποίηση μίας $\mathcal{EL}^+$ οντολογίας $\mathcal{O}$ μπορεί να γίνει σε δύο φάσεις:

1. Εφαρμόζοντας εξαντλητικά τους κανόνες NF1 έως NF3.
2. Εφαρμόζοντας εξαντλητικά τους κανόνες NF4 έως NF6.

Με τον όρο "εφαρμογή κανόνα" εννοούμε την αντικατάσταση των αξιωμάτων της οντολογίας με μορφή ίδια με αυτή του αριστερού μέλους ενός κανόνα με τα αξιώματα του δεξιού μέλους.

5.1.2 Αφηρημένος Αλγόριθμος Υπολογισμού Αντιστοιχίσεων

Στην ενότητα αυτή υποθέτουμε χωρίς βλάβη της γενικότητας ότι η οντολογία εισόδου $\mathcal{O}$ ευρίσκεται σε κανονική μορφή. Έστω ότι $RN_{\mathcal{O}}$ είναι το σύνολο όλων των ονομάτων ρόλων που εμφανίζονται στην $\mathcal{O}$. Ο αλγόριθμος υπολογίζει:

1. μία αντιστοίχηση (mapping) S που αναθέτει σε κάθε στοιχείο $A \in CN_{\mathcal{O}}$ ένα υποσύνολο $S(A)$ του $CN_{\mathcal{O}}^{\top}$ και
2. μία αντιστοίχηση R που αναθέτει σε κάθε στοιχείο $r \in RN_{\mathcal{O}}$ μία διμελή σχέση $R(r)$ στο $CN_{\mathcal{O}}^{\top}$.

Αυτές οι αντιστοιχίσεις αναδεικνύουν τις υπονοούμενες σχέσεις υπαγωγής, καθώς:

1. $B \in S(A)$ συνεπάγεται ότι $A \sqsubseteq_{\mathcal{O}} B$ και
2. $(A, B) \in R(r)$ συνεπάγεται ότι $A \sqsubseteq_{\mathcal{O}} \exists r.B$.

Τα σύνολα αντιστοιχίσεων αρχικοποιούνται θέτοντας $S(A) := \{A, \top\}$ για κάθε $A \in CN_{\mathcal{O}}$ και $R(r) := \emptyset$ για κάθε $r \in RN_{\mathcal{O}}$. Εν συνεχεία τα σύνολα $S(A)$ και $R(r)$ επεκτείνονται εφαρμόζοντας τους κανόνες συμπλήρωσης που αναφέρονται στον **Πίνακα 5**, μέχρις ότου δε μπορεί να εφαρμοστεί κανένας άλλος κανόνας.

Στο [8] έχει αποδειχθεί ότι ο αλγόριθμος αυτός είναι συνεπής και πλήρης, καθώς μετά τον τερματισμό ισχύει $B \in S(A)$ αν και μόνο αν $A \sqsubseteq_{\mathcal{O}} B$, για κάθε $A, B \in CN_{\mathcal{O}}^{\top}$. Έχει επίσης αποδειχθεί ότι ο αλγόριθμος πάντοτε τερματίζει σε πολυωνυμικό χρόνο ως προς το μέγεθος της οντολογίας εισόδου.

Κανονική Μορφή		**Κανόνας Συμπλήρωσης**
$A_1 \sqcap ... \sqcap A_n \sqsubseteq B$	R1	**Αν** $A_1, ..., A_n \in S(X)$, $A_1 \sqcap ... \sqcap A_n \sqsubseteq B \in \mathcal{O}$ και $B \notin S(X)$ **τότε** $S(X) := S(X) \cup \{B\}$.
$A \sqsubseteq \exists r.B$	R2	**Αν** $A \in S(X)$, $A \sqsubseteq \exists r.B \in \mathcal{O}$ και $(X, B) \notin R(r)$ **τότε** $R(r) := R(r) \cup \{(X, B)\}$.
$\exists r.A \sqsubseteq B$	R3	**Αν** $(X, Y) \in R(r)$, $A \in S(Y)$, $\exists r.A \sqsubseteq B \in \mathcal{O}$ και $B \notin S(X)$ **τότε** $S(X) := S(X) \cup \{B\}$.
$r \sqsubseteq s$	R4	**Αν** $(X, Y) \in R(r)$, $r \sqsubseteq s \in \mathcal{O}$ και $(X, Y) \notin R(s)$ **τότε** $R(s) := R(s) \cup \{(X, Y)\}$.
$r \circ s \sqsubseteq t$	R5	**Αν** $(X, Y) \in R(r)$, $(Y, Z) \in R(s)$, $r \circ s \sqsubseteq t \in \mathcal{O}$ και $(X, Z) \notin R(t)$ **τότε** $R(t) := R(t) \cup \{(X, Z)\}$.

Πίνακας 5: *Αξιώματα κανονικής μορφής και κανόνες συμπλήρωσης*

5.1.3 Βελτιωμένος Αλγόριθμος Υπολογισμού Αντιστοιχίσεων

Ένα από τα βασικότερα προβλήματα που συναντάται κατά την υλοποίηση του αλγορίθμου που περιεγράφη στην προηγούμενη ενότητα είναι η ανάπτυξη ενός αποδοτικού τρόπου εύρεσης του επόμενου κανόνα που πρέπει να εφαρμοστεί. Αν η αναζήτηση για τον επόμενο κανόνα γίνει με αλγόριθμο εξαντλητικής αναζήτησης – ωμής βίας, δεν μπορούμε να αναμένουμε ικανοποιητικό χρόνο εκτέλεσης για μεγάλες οντολογίες εισόδου. Ως λύση στο πρόβλημα αυτό προτείνεται η ακόλουθη βελτιωμένη έκδοση του αλγορίθμου υπολογισμού των αντιστοιχίσεων, η οποία ενεπνεύσθη από τον γραμμικού χρόνου αλγόριθμο ελέγχου ικανοποιησιμότητας προτάσεων Horn ([14]). Η βελτιωμένη αυτή έκδοση του αλγορίθμου χρησιμοποιεί ένα σύνολο ουρών, μία για κάθε ονοματική έννοια που συναντάται στην οντολογία εισόδου, προκειμένου να καθοδηγηθεί στην εφαρμογή των κανόνων συμπλήρωσης. Διαισθητικά, η ουρά περιέχει τις τροποποιήσεις που οφείλεται να γίνουν στα σύνολα $S(A)$ και $R(r)$. Οι πιθανές εγγραφές μίας ουράς είναι της μορφής

$$B_1 \sqcap ... \sqcap B_n \rightarrow B' \text{ και } \exists r.B,$$

όπου $B_1, ..., B_n$, B και B' είναι ονόματα εννοιών, r όνομα ρόλου και $n \geq 0$. Για την περίπτωση $n = 0$ η εγγραφή τύπου $B_1 \sqcap ... \sqcap B_n \rightarrow B'$ της ουράς γράφεται απλά ως B'. Επίσης:

1. μία εγγραφή τύπου $B_1 \sqcap ... \sqcap B_n \rightarrow B'$ στην $queue(A)$ σημαίνει ότι η έννοια B' πρέπει να εισαχθεί στο $S(A)$, αν αυτό ήδη περιέχει τα $B_1, ..., B_n$, και
2. μία εγγραφή τύπου $\exists r.B$ στην $queue(A)$ σημαίνει ότι η δυάδα (A, B) θα πρέπει να εισαχθεί στο $R(r)$.

Το γεγονός ότι μία τέτοια προσθήκη μπορεί να ενεργοποιήσει άλλους κανόνες πρέπει να ληφθεί υπ' όψιν επεκτείνοντας καταλλήλως τις ουρές όποτε λαμβάνει χώρα η προσθήκη.

Επίσης, θεωρούμε μία αντιστοίχηση $\hat{\mathcal{O}}$ από ονόματα εννοιών σε σύνολα από εγγραφές ουρών, βάσει μίας κανονικοποιημένης οντολογίας εισόδου $\mathcal{O}$, ως εξής:

1. Αν $A_1 \sqcap ... \sqcap A_n \sqsubseteq B \in \mathcal{O}$ και $A = A_i$, τότε
$$A_1 \sqcap ... \sqcap A_{i-1} \sqcap A_{i+1} \sqcap ... \sqcap A_n \rightarrow B \in \hat{\mathcal{O}}(A).$$
2. Αν $A \sqsubseteq \exists r.B \in \mathcal{O}$, τότε
$$\exists r.B \in \hat{\mathcal{O}}(A).$$

Αντιστοίχως, για κάθε έννοια $\exists r.A$ συμβολίζεται ως $\hat{\mathcal{O}}(\exists r.A)$ το ελάχιστο σύνολο εγγραφών ουράς τέτοιο ώστε αν $\exists r.A \sqsubseteq B \in \mathcal{O}$, τότε $B \in \hat{\mathcal{O}}(\exists r.A)$.

Στον βελτιωμένο αλγόριθμο η $queue(A)$ αρχικοποιείται με $\hat{\mathcal{O}}(A) \cup \hat{\mathcal{O}}(\top)$ – αφού το σύνολο $S(A)$ αρχικοποιείται με $\{A, \top\}$ –. Αυτό σημαίνει πως εισάγουμε στην $queue(A)$ τις άμεσες συνέπειες του να είναι το A ένα στιγμιότυπο του εαυτού του και του $\top$. Στη συνέχεια αφαιρούμε επαναληπτικά εγγραφές από τις ουρές και τις επεξεργαζόμαστε με χρήση της διαδικασίας *επεξεργασία* (Αλγόριθμος 1). Συγκεκριμένα, η διαδικασία *επεξεργασία*(A, X) καλείται όταν η ουρά της έννοιας A δεν ήταν κενή και εξήχθη από την $queue(A)$ το στοιχείο X προκειμένου να επεξεργαστεί. Η διαδικασία *επεξεργασία−νέας−ακμής*(A, r, B) (Αλγόριθμος 2) καλείται από την *επεξεργασία* προκειμένου να διαχειριστεί της συνέπειες της προσθήκης ενός νέου ζεύγους (A, B) στο $R(r)$.

Διαδικασία επεξεργασία(A, X)

Αν $X = B_1 \sqcap ... \sqcap B_n \rightarrow B'$ **και** $B \notin S(A)$ **τότε**

Αν $\{B_1, ..., B_n\} \subseteq S(A)$ **τότε**

$S(A) := S(A) \cup \{B\}$

$queue(A) := queue(A) \cup \hat{\mathcal{O}}(B)$

Για όλα τα ονόματα εννοιών A' **και ονόματα ρόλων** r **όπου**

$(A', A) \in R(r)$

$queue(A') := queue(A') \cup \hat{\mathcal{O}}(\exists r.B)$

Αν $X = \exists r.B$ **και** $(A, B) \notin R(r)$ **τότε**

επεξεργασία-νέας-ακμής(A, r, B)

Τέλος Διαδικασίας

Αλγόριθμος 1: *Αλγόριθμος Επεξεργασίας Ουρών: επεξεργασία(A, X)*

Με το σύμβολο $\sqsubseteq^*_{\mathcal{O}}$ συμβολίζουμε το ανακλαστικό – μεταβατικό κλείσιμο των αξιωμάτων ιεραρχίας ρόλων της $\mathcal{O}$. Η επεξεργασία των ουρών συνεχίζει μέχρις ότου όλες οι ουρές αδειάσουν, ενώ μπορεί να παρατηρηθεί ότι ο βελτιωμένος αυτός αλγόριθμος δεν απαιτείται να διενεργεί κανέναν έλεγχο σχετικά με το ποιος κανόνας συμπλήρωσης πρέπει να εφαρμοστεί στη συνέχεια.

Διαδικασία επεξεργασία-νέας-ακμής(A, r, B)

Για όλα τα ονόματα ρόλων s **όπου** $r \sqsubseteq^*_{\mathcal{O}} s$

$R(s) := R(s) \cup \{(A, B)\}$

$queue(A) := queue(A) \cup \bigcup_{\{B' | B' \in S(B)\}} \hat{\mathcal{O}}(\exists s.B')$

Για όλα τα ονόματα εννοιών A' **και ονόματα ρόλων** t, u **όπου**

$t \circ s \sqsubseteq u \in \mathcal{O}$ **και** $(A', A) \in R(t)$ **και** $(A', B) \notin R(u)$

επεξεργασία-νέας-ακμής(A', u, B)

Για όλα τα ονόματα εννοιών B' **και ονόματα ρόλων** t, u **όπου**

$s \circ t \sqsubseteq u \in \mathcal{O}$ **και** $(B, B') \in R(t)$ **και** $(A, B') \notin R(u)$

επεξεργασία-νέας-ακμής(A, u, B')

Τέλος Διαδικασίας

Αλγόριθμος 2: *Αλγόριθμος Επεξεργασίας Ουρών: επεξεργασία-νέας-ακμής(A, r, B)*

Ο παραπάνω βελτιωμένος αλγόριθμος εκτελείται σε πολυωνυμικό χρόνο ($O(n^4)$) και είναι συνεπής και πλήρης, καθώς μετά τον τερματισμό ισχύει $B \in S(A)$ αν και μόνο αν $A \sqsubseteq_{\mathcal{O}} B$, για κάθε $A, B \in CN_{\mathcal{O}}^{\top}$. Απόδειξη αυτού μπορεί να βρεθεί στο [14].

5.1.4 Υπολογισμός Άμεσων Σχέσεων Υπαγωγής

Το αποτέλεσμα της προηγούμενης διαδικασίας είναι τα υπολογισθέντα σύνολα S και R για κάθε ονοματική έννοια και ρόλο της οντολογίας. Τα σύνολα S ονομάζονται **σύνολα υπαγουσών εννοιών (subsumer sets)** και περιέχουν όλες τις υπερκλάσεις μίας έννοιας. Όμως, αυτή η μορφή των αποτελεσμάτων της διαδικασίας συλλογιστικής δεν είναι πάντα η επιθυμητή και γι' αυτό το λόγο πολλές μηχανές συλλογιστικής υλοποιούν και ένα επιπλέον στάδιο υπολογισμών, στο οποίο και υπολογίζουν τις άμεσες σχέσεις υπαγωγής, δηλαδή το ποιες κλάσεις υπάγουν άμεσα μία έννοια, ποιες είναι ισοδύναμές της (δηλαδή την υπάγουν και υπάγονται από αυτή) και ποιες υπάγονται άμεσα.

Παραδείγματος χάριν, για μία έννοια A το σύνολο των εννοιών B που την υπάγουν άμεσα πληροί τις ακόλουθες ιδιότητες:

1. $A \sqsubseteq_{\mathcal{O}} B$
2. $\nexists B' \notin \{A, B\} | A \sqsubseteq_{\mathcal{O}} B' \sqsubseteq_{\mathcal{O}} B$

Ενώ οι αντίστοιχες ιδιότητες που πληροί μία έννοια A η οποία υπάγει άμεσα ένα σύνολο εννοιών B είναι:

1. $B \sqsubseteq_{\mathcal{O}} A$
2. $\nexists B' \notin \{A, B\} | B \sqsubseteq_{\mathcal{O}} B' \sqsubseteq_{\mathcal{O}} A$

Τα σύνολα αυτά των εννοιών δημιουργούν έναν **συνεκτικό ακυκλικό γράφο υπαγωγής (subsumption Directed Acyclic Graph - DAG)**, ο οποίος αναπαριστά τις άμεσες σχέσεις μεταξύ εννοιών σε ολόκληρη την οντολογία.

Στο σημείο αυτό πρέπει να σημειωθεί πως τα διάφορα σύνολα υπαγόντων και υπαγομένων δεν αναφέρονται μόνο σε συγκεκριμένες έννοιες, αλλά αφορούν ολόκληρες τις κλάσεις ισοδυναμίας στην οποία οι έννοιες αυτές ανήκουν. Τα πλεονεκτήματα της χρήσης ενός συνεκτικού ακυκλικού γράφου υπαγωγής σε σύγκριση με τα σύνολα υπαγουσών εννοιών (subsumer sets) είναι το γεγονός ότι ο γράφος αποτελεί πιο συνεπτυγμένη μορφή αναπαράστασης των δεδομένων (λιγότερος πλεονασμός κατά την αποθήκευση), ενώ επίσης υποστηρίζει πολύ εύκολα την περιδιάβαση (browsing) της ιεραρχίας υπαγωγών, καθώς από μία έννοια μπορούμε να επισκεφτούμε άμεσα τις υπαγόμενες απ' αυτήν και αυτές που την υπάγουν. Το βασικό μειονέκτημα είναι το ότι η εξέταση της αληθείας του ερωτήματος $A \sqsubseteq_{\mathcal{O}}^{?} B$ απαιτεί την εξέταση της υπάρξεως μονοπατιού (reachability) στο γράφο από το B στο A, αντί ενός απλού ελέγχου στο σύνολο $S(A)$ των υπερκλάσεων της έννοιας A. Βεβαίως, αυτό το πρόβλημα παρακάμπτεται αν ο αλγόριθμος μπορεί να διατηρήσει και τις δύο δομές αποθήκευσης, δηλαδή τον γράφο και τα σύνολα S, στη μνήμη.

Ένας τρόπος υπολογισμού του συνεκτικού ακυκλικού γράφου υπαγωγής είναι μέσω του υπολογισμού, για κάθε όνομα έννοιας A, των ακολούθων συνόλων:

1. του συνόλου $SS(A) := \{B \in S(A) | A \notin S(B)\}$, το οποίο περιλαμβάνει τις έννοιες που υπάγουν αυστηρά την A, δηλαδή τις έννοιες που υπάγουν την A και δεν είναι ισοδύναμές της,

2. του συνόλου $DS(A) := SS(A) \setminus (\bigcup_{B \in SS(A)} SS(B))$ των εννοιών που υπάγουν άμεσα την A και
3. του συνόλου $DS^{-}(A) := \{B | A \in DS(B)\}$ των άμεσα υπαγομένων από την Α εννοιών.

Τα σύνολα $DS(A)$ και $DS^{-}(A)$ αποτελούν μία αναπαράσταση του γράφου υπαγωγής. Παρ' όλα αυτά, ο τρόπος αυτός κατασκευής του γράφου υπαγωγής δεν είναι ο αποδοτικότερος: η κατασκευή του $DS^{-}(A)$ απαιτεί χρόνο $O(n^2)$, γι' αυτό και χρησιμοποιείται ένας διαφορετικός αλγόριθμος ([14]).

Για να εξηγηθεί η βασική ιδέα του αλγορίθμου που χρησιμοποιήθηκε, ας υποθέσουμε ότι έχουμε ήδη υπολογίσει ένα τμήμα του γράφου υπαγωγής για κάποιο υποσύνολο των ονομάτων εννοιών και είμαστε έτοιμοι να εισάγουμε την έννοια A στον γράφο. Ξεκινάμε υπολογίζοντας το σύνολο $SS(A)$ των εννοιών που υπάγουν αυστηρά την A. Τα στοιχεία του $S(A) \setminus SS(A)$ αποτελούν τις ισοδύναμες της A έννοιες. Για να βρούμε τις έννοιες που υπάγουν αυστηρά την A ανάμεσα στο $SS(A)$, προχωρούμε ως εξής:

1. Αν όλα τα στοιχεία του $SS(A)$ ανήκουν στον ήδη υπολογισθέντα γράφο, τότε μπορούμε να βρούμε τις έννοιες που υπάγουν άμεσα την A χρησιμοποιώντας ένα απλό αλγόριθμο διάσχισης γράφου για να "σημειώσουμε" όλες τις έννοιες του γράφου που υπάγουν αυστηρά στοιχεία του $SS(A)$. Οι έννοιες που αναζητούμε τότε είναι όλα τα στοιχεία του $SS(A)$ που δεν έχουν "σημειωθεί".
2. Αν υπάρχουν στοιχεία του $SS(A)$ τα οποία δεν ανήκουν στον μέχρι στιγμής υπολογισθέντα γράφο, τότε πρέπει πρώτα να εισάγουμε στο γράφο αυτά τα στοιχεία (με αναδρομικές κλήσεις της συνάρτησης εισαγωγής) προτού εισάγουμε την A. Με τον τρόπο αυτό είμαστε βέβαιοι ότι όταν εισαχθεί το όνομα μίας έννοιας στον γράφο τότε τη στιγμή εκείνη όλες οι έννοιες οι οποίες την υπάγουν θα βρίσκονται ήδη στον γράφο, ενώ σ' αυτόν δε θα βρίσκεται καμία έννοια που υπάγεται από αυτήν. Συνεπώς, ο αλγόριθμος αυτός δε χρειάζεται να υπολογίζει τις άμεσα υπαγόμενες έννοιες, αλλά αρκεί να επεκταθεί το σύνολο των άμεσα υπαγόμενων από την έννοια B εισάγοντας την έννοια A, εάν βρεθεί ότι η Bυπάγει άμεσα την A.

Στα πλαίσια 3, 4 και 5 περιέχεται η αναπαράσταση του παραπάνω αλγορίθμου σε ψευδοκώδικα. Το σύνολο *γονείς*(A) χρησιμοποιείται για να αποθηκεύει τις έννοιες που υπάγουν άμεσα την A, το σύνολο *παιδιά*(A) περιέχει τις άμεσα υπαγόμενες από την A έννοιες, ενώ τέλος το σύνολο *ισοδύναμες*(A) περιέχει τις ισοδύναμες της A έννοιες. Η περιγραφή του αλγορίθμου χρειάζεται προσοχή, καθώς δε φανερώνει άμεσα την ήδη αναφερθείσα διαφορά ανάμεσα στο όνομα μίας έννοιας και σ' έναν κόμβο του γράφου: ο κόμβος του γράφου στην ουσία αποτελεί κλάση ισοδυναμίας εννοιών. Αυτό έχει σαν αποτέλεσμα διάφορες λειτουργίες να μην υλοποιούνται απ' ευθείας, αλλά να πρέπει κάθε φορά να ενημερωθούν όλες οι ισοδύναμες έννοιες. Για παράδειγμα, αν εισάγουμε στο σύνολο *γονείς*(A) μία έννοια, τότε αυτή πρέπει να εισαχθεί και στα σύνολα *γονείς* όλων των ισοδυνάμων της A εννοιών.

Διαδικασία $\upsilon\pi o\lambda o\gamma\iota\sigma\mu\acute{o}\varsigma - \Sigma A\Gamma()$

Για όλα τα ονόματα εννοιών $X \in CN_{\mathcal{O}}^{\top}$

$\kappa\alpha\tau\alpha\tau\alpha\chi\theta\epsilon\acute{\iota}\sigma\alpha(X) := \psi\epsilon\upsilon\delta\acute{\epsilon}\varsigma$

$\gamma o\nu\epsilon\acute{\iota}\varsigma(X) := \pi\alpha\iota\delta\iota\acute{\alpha}(X) := \iota\sigma o\delta\acute{\upsilon}\nu\alpha\mu\epsilon\varsigma(X) := \emptyset$

Για κάθε όνομα έννοιας $A \in CN_{\mathcal{O}}^{\top}$

Αν δεν ισχύει ότι $\tau\alpha\xi\iota\nu o\mu\eta\mu\acute{\epsilon}\nu\eta(A)$ **τότε**

$\Sigma A\Gamma - \tau\alpha\xi\iota\nu\acute{o}\mu\eta\sigma\eta(A)$

Τέλος Διαδικασίας

Αλγόριθμος 3: *Αλγόριθμος υπολογισμού ΣΑΓ υπαγωγής: υπολογισμός-ΣΑΓ()*

Διαδικασία $\Sigma A\Gamma - \tau\alpha\xi\iota\nu\acute{o}\mu\eta\sigma\eta(A)$

$\upsilon\pi o\psi\acute{\eta}\varphi\iota o\iota := \emptyset$

Για κάθε $B \in S(A)$

Αν $A \in S(B)$ **τότε**

$\tau\alpha\xi\iota\nu o\mu\eta\mu\acute{\epsilon}\nu\eta(B) := \alpha\lambda\eta\theta\acute{\epsilon}\varsigma$

$\iota\sigma o\delta\acute{\upsilon}\nu\alpha\mu\epsilon\varsigma(A) := \iota\sigma o\delta\acute{\upsilon}\nu\alpha\mu\epsilon\varsigma(A) \cup \{B\}$

αλλιώς

Αν δεν ισχύει ότι $\tau\alpha\xi\iota\nu o\mu\eta\mu\acute{\epsilon}\nu\eta(B)$ **τότε**

$\Sigma A\Gamma - \tau\alpha\xi\iota\nu\acute{o}\mu\eta\sigma\eta(B)$

$\upsilon\pi o\psi\acute{\eta}\varphi\iota o\iota := \upsilon\pi o\psi\acute{\eta}\varphi\iota o\iota \cup \{B\}$

$\Sigma A\Gamma - \epsilon\iota\sigma\alpha\gamma\omega\gamma\acute{\eta}(A, \upsilon\pi o\psi\acute{\eta}\varphi\iota o\iota)$

$\tau\alpha\xi\iota\nu o\mu\eta\mu\acute{\epsilon}\nu\eta(A) := \alpha\lambda\eta\theta\acute{\epsilon}\varsigma$

Τέλος Διαδικασίας

Αλγόριθμος 4: *Αλγόριθμος υπολογισμού ΣΑΓ υπαγωγής: ΣΑΓ-ταξινόμηση(A)*

Διαδικασία $\Sigma A\Gamma - \varepsilon\iota\sigma\alpha\gamma\omega\gamma\acute{\eta}(A, \upsilon\pi o\psi\acute{\eta}\varphi\iota o\iota)$

Για κάθε $X \in CN_{\mathcal{O}}^{\top}$

$\sigma\eta\mu\varepsilon\iota\omega\mu\acute{\varepsilon}\nu o(X) := \psi\varepsilon\upsilon\delta\acute{\varepsilon}\varsigma$

Για κάθε $B \in \upsilon\pi o\psi\acute{\eta}\varphi\iota o\iota$

Για κάθε $X \in \gamma o\nu\varepsilon\acute{\iota}\varsigma(B)$

$\sigma\eta\mu\varepsilon\iota\omega\mu\acute{\varepsilon}\nu o(X) := \alpha\lambda\eta\theta\acute{\varepsilon}\varsigma$

'Οσο υπάρχουν $X, Y \in CN_{\mathcal{O}}^{\top}$ **τέτοια ώστε** $\sigma\eta\mu\varepsilon\iota\omega\mu\acute{\varepsilon}\nu o(X)$, $Y \in \gamma o\nu\varepsilon\acute{\iota}\varsigma(X)$ **και δεν ισχύει ότι** $\sigma\eta\mu\varepsilon\iota\omega\mu\acute{\varepsilon}\nu o(Y)$ **επανάλαβε**

$\sigma\eta\mu\varepsilon\iota\omega\mu\acute{\varepsilon}\nu o(Y) := \alpha\lambda\eta\theta\acute{\varepsilon}\varsigma$

$\gamma o\nu\varepsilon\acute{\iota}\varsigma(A) := \gamma o\nu\varepsilon\acute{\iota}\varsigma(A) \cup \{B \in \upsilon\pi o\psi\acute{\eta}\varphi\iota o\iota : \sigma\eta\mu\varepsilon\iota\omega\mu\acute{\varepsilon}\nu o(B) = \psi\varepsilon\upsilon\delta\acute{\varepsilon}\varsigma\}$

Για κάθε $B \in \gamma o\nu\varepsilon\acute{\iota}\varsigma(A)$

$\pi\alpha\iota\delta\iota\acute{\alpha}(B) := \pi\alpha\iota\delta\iota\acute{\alpha}(B) \cup \{A\}$

Τέλος Διαδικασίας

Αλγόριθμος 5: *Αλγόριθμος υπολογισμού ΣΑΓ υπαγωγής: ΣΑΓ-εισαγωγή(Α, υποψήφιοι)*

5.2 Ταξινόμηση Οντολογίας f-$\mathcal{EL}^+$

Όπως συνέβη και με τον σαφούς λογικής αλγόριθμο ταξινόμησης για την $\mathcal{EL}^+$, πρώτα θα παρουσιαστεί ένας αφηρημένος αλγόριθμος ταξινόμησης (classification) πολυωνυμικού χρόνου για την περιγραφική $f - \mathcal{EL}^+$, ενώ στη συνέχεια θα παρουσιαστεί μία βελτιωμένη έκδοση του ιδίου αλγορίθμου που επιτυγχάνει χαμηλότερη θεωρητική χρονική πολυπλοκότητα – αν και αυτή παραμένει πολυωνυμική – και συνεπώς υψηλότερη απόδοση. Και οι δύο αυτοί αλγόριθμοι (αφηρημένος και συγκεκριμένος) παρουσιάζονται στο [12].

Τόσο σε συστήματα συλλογιστικής Περιγραφικών Λογικών βασιζόμενα σε αλγορίθμους tableau, όσο και στα αρχικά συστήματα βασιζόμενα σε αλγορίθμους δομικής υπαγωγής, η ιεραρχία υπαγωγών υπολογίζεται εκτελώντας πολλαπλούς ελέγχους υπαγωγής εννοιών. Αυτό σημαίνει ότι σε τέτοια συστήματα υπάρχει ανάγκη όχι μόνο βελτιστοποίησης της διαδικασίας ελέγχου της ισχύος μίας υπαγωγής εννοιών, αλλά και περιορισμού του πλήθους των ελέγχων υπαγωγής που απαιτούνται προκειμένου να υπολογισθεί η συνολική ιεραρχία. Αντιθέτως, ο αλγόριθμος υπαγωγής που παρουσιάστηκε στο [12] υπολογίζει ταυτόχρονα τις σχέσεις υπαγωγής μεταξύ όλων των ζευγών ονομάτων εννοιών της οντολογίας εισόδου.

Στο σημείο αυτό υπενθυμίζεται πως όλα τα στάδια του ήδη περιγραφέντος αλγορίθμου για την $\mathcal{EL}^+$ μπορούν να ιδωθούν ως ειδικές περιπτώσεις του γενικού αλγορίθμου ασαφούς λογικής: συγκεκριμένα, κάθε αξίωμα υπαγωγής της $\mathcal{EL}^+$ μπορεί να θεωρηθεί ισοδύναμο με το ίδιο αξίωμα της $f - \mathcal{EL}^+$ το οποίο ισχύει με βαθμό βεβαιότητας 1.

5.2.1 Μετατροπή σε Κανονική Μορφή Οντολογιών f-$\mathcal{EL}^+$

Προτού περιγραφεί ο πολυωνυμικού χρόνου αλγόριθμος ταξινόμησης για την $f-\mathcal{EL}^+$ πρέπει να ορισθεί μία κατάλληλη κανονική μορφή για τα αξιώματα της οντολογίας ([14]).

Γενικά, σε μία οντολογία $\mathcal{O}$, ορίζουμε με τους συμβολισμούς $CN_{\mathcal{O}}^{\top}$ και $CN_{\mathcal{O}}$ τα σύνολα των ονομάτων εννοιών που εμφανίζονται στην $\mathcal{O}$, συμπεριλαμβανομένης ή μη συμπεριλαμβανομένης της έννοιας top ($\top$) αντίστοιχα.

Λέμε λοιπόν ότι η $\mathcal{O}$ είναι σε κανονική μορφή εάν:

3. Όλα τα γενικευμένα αξιώματα υπαγωγής εννοιών (GCIs) της οντολογίας λαμβάνουν μία εκ των ακολούθων μορφών, όπου $A_i \in CN_{\mathcal{O}}^{\top}$ και $B \in CN_{\mathcal{O}}$:

$$\langle A_1 \sqcap ... \sqcap A_n \sqsubseteq B, n\rangle,$$

$$\langle A_1 \sqsubseteq \exists r.A_2, n\rangle \text{ και}$$

$$\langle \exists r.A_1 \sqsubseteq B, n\rangle.$$

4. Όλα τα αξιώματα υπαγωγής ρόλων (RIs) είναι της μορφής:

$$r \sqsubseteq s \text{ ή } r_1 \circ r_2 \sqsubseteq s.$$

Όπως ακριβώς και κάθε $\mathcal{EL}^+$ οντολογία μπορεί να μετασχηματιστεί σε μία **κανονικοποιημένη** οντολογία $\mathcal{O}'$, η οποία να είναι **συντηρητική επέκταση** της $\mathcal{O}$, εισάγοντας νέα ονόματα ρόλων και εννοιών, έτσι ακριβώς συμβαίνει και για την $f-\mathcal{EL}^+$. Με τον όρο συντηρητική επέκταση εννοούμε πως αν $\langle C \sqsubseteq_{\mathcal{O}} D, n\rangle$ τότε και μόνο τότε $\langle C \sqsubseteq_{\mathcal{O}'} D, n\rangle$ για όλες τις περιγραφές εννοιών C, D που μπορούν να κατασκευαστούν από τα ονόματα εννοιών και ρόλων που συναντώνται στην $\mathcal{O}$. Μάλιστα, αποδεικνύεται ότι **ο μετασχηματισμός αυτός μπορεί να λάβει χώρα σε γραμμικό χρόνο** ως προς το μέγεθος της οντολογίας. Μία λεπτομερής απόδειξη είναι δυνατό να βρεθεί στο [14].

Κανόνας Κανονικοποίησης	**Αντικατάσταση**
NF1	$r_1 \circ ... \circ r_k \sqsubseteq s \rightsquigarrow r_1 \circ ... \circ r_{k-1} \sqsubseteq u, u \circ r_k \sqsubseteq s$
NF2	$\langle C_1 \sqcap ... \sqcap \hat{C} \sqcap ... \sqcap C_n \sqsubseteq D, n\rangle \rightsquigarrow$ $\langle \hat{C} \sqsubseteq A, n\rangle, \langle C_1 \sqcap ... \sqcap A \sqcap ... \sqcap C_n \sqsubseteq D, n\rangle$
NF3	$\langle \exists r.\hat{C} \sqsubseteq D, n\rangle \rightsquigarrow \langle \hat{C} \sqsubseteq A, n\rangle, \langle \exists r.A \sqsubseteq D, n\rangle$
NF4	$\langle \hat{C} \sqsubseteq \hat{D}, n\rangle \rightsquigarrow \langle \hat{C} \sqsubseteq A, n\rangle, \langle A \sqsubseteq \hat{D}, n\rangle$
NF5	$\langle B \sqsubseteq \exists r.\hat{C}, n\rangle \rightsquigarrow \langle B \sqsubseteq \exists r.A, n\rangle, \langle A \sqsubseteq \hat{C}, n\rangle$
NF6	$\langle B \sqsubseteq C \sqcap D\rangle \rightsquigarrow \langle B \sqsubseteq C, n\rangle, \langle B \sqsubseteq D, n\rangle$

Πίνακας 6: *Κανόνες κανονικοποίησης οντολογιών f-EL⁺*

Η κανονικοποίηση μίας $f-\mathcal{EL}^+$ οντολογίας $\mathcal{O}$ μπορεί να γίνει σε δύο φάσεις:

1. Εφαρμόζοντας εξαντλητικά τους κανόνες NF1 έως NF3.

2. Εφαρμόζοντας εξαντλητικά τους κανόνες NF4 έως NF6.

Με τον όρο "εφαρμογή κανόνα" εννοούμε την αντικατάσταση των αξιωμάτων της οντολογίας με μορφή ίδια με αυτή του αριστερού μέλους ενός κανόνα με τα αξιώματα του δεξιού μέλους.

Αποδεικνύεται ότι η υπαγωγή εννοιών με βάση $f - \mathcal{EL}^+$ οντολογίες μπορεί να αναχθεί σε πολυωνυμικό χρόνο σε υπαγωγή εννοιών με βάση κανονικοποιημένες $f - \mathcal{EL}^+$ οντολογίες, δηλαδή η κανονικοποίηση απαιτεί πολυωνυμικό χρόνο. Η προσέγγιση που ακολουθείται είναι αντίστοιχη με αυτή στο [14].

5.2.2 Αφηρημένος Αλγόριθμος Υπολογισμού Αντιστοιχίσεων

Στην ενότητα αυτή υποθέτουμε χωρίς βλάβη της γενικότητας ότι η οντολογία εισόδου $\mathcal{O}$ ευρίσκεται σε κανονική μορφή. Έστω ότι $RN_{\mathcal{O}}$ είναι το σύνολο όλων των ονοματικών ρόλων που εμφανίζονται στην $\mathcal{O}$. Ο αλγόριθμος υπαγωγής για τις κανονικοποιημένες $f - \mathcal{EL}^+$ μπορεί να περιορισθεί σε έλεγχο υπαγωγής μεταξύ ονοματικών εννοιών και όχι γενικευμένων εννοιών. Αυτό συμβαίνει γιατί:

$\langle C \sqsubseteq_{\mathcal{O}} D, n\rangle$ αν και μόνο αν $\langle A \sqsubseteq_{\mathcal{O}'} B, n\rangle$, όπου $\mathcal{O}' = \mathcal{O} \cup \{\langle A \sqsubseteq C, n\rangle, \langle D \sqsubseteq B, n\rangle\}$, όπου A, B νέα ονόματα εννοιών.

Ο αλγόριθμος λοιπόν υπολογίζει:

1. μία αντιστοίχηση (mapping) S που αναθέτει σε κάθε στοιχείο $A \in CN_{\mathcal{O}}$ ένα υποσύνολο $S(A)$ του $CN_{\mathcal{O}}^{\top} \times [0,1]$ και
2. μία αντιστοίχηση R που αναθέτει σε κάθε στοιχείο $r \in RN_{\mathcal{O}}$ μία τριμελή σχέση $R(r)$, υποσύνολο του $CN_{\mathcal{O}}^{\top} \times CN_{\mathcal{O}}^{\top} \times [0,1]$.

Στο σημείο αυτό γίνεται φανερό πως εξ' αιτίας των ασαφών σχέσεων υπαγωγής οι αντιστοιχίσεις S και R έχουν επεκταθεί και εκτείνονται σε υποσύνολα των $CN_{\mathcal{O}}^{\top} \times [0,1]$ και $CN_{\mathcal{O}}^{\top} \times CN_{\mathcal{O}}^{\top} \times [0,1]$ αντίστοιχα. Αυτές οι αντιστοιχίσεις αναδεικνύουν τις υπονοούμενες σχέσεις υπαγωγής, καθώς:

1. $\langle B, n\rangle \in S(A)$ συνεπάγεται ότι $\langle A \sqsubseteq_{\mathcal{O}} B, n\rangle$ και
2. $\langle A, B, n\rangle \in R(r)$ συνεπάγεται ότι $\langle A \sqsubseteq_{\mathcal{O}} \exists r.B, n\rangle$.

Τα σύνολα αντιστοιχίσεων αρχικοποιούνται θέτοντας $S(A) := \{\langle A, 1\rangle, \langle \top, 1\rangle\}$ για κάθε $A \in CN_{\mathcal{O}}$ - αφού κάθε έννοια υπάγεται με βαθμό ίσο με τη μονάδα στον εαυτό της και στο $\top$ – και $R(r) := \emptyset$ για κάθε $r \in RN_{\mathcal{O}}$. Εν συνεχεία τα σύνολα $S(A)$ και $R(r)$ επεκτείνονται εφαρμόζοντας τους κανόνες συμπλήρωσης που αναφέρονται στον **Πίνακα 7**, μέχρις ότου δε μπορεί να εφαρμοστεί κανένας άλλος κανόνας.

Αντίστοιχα με την προσέγγιση που ακολουθείται στο [8] για την $\mathcal{EL}^+$, μπορεί να αποδειχθεί ότι ο αλγόριθμος αυτός είναι συνεπής και πλήρης, καθώς μετά τον τερματισμό ισχύει $\langle B, n\rangle \in S(A)$ αν και μόνο αν $\langle A \sqsubseteq_{\mathcal{O}} B, n'\rangle$, για κάθε $A, B \in CN_{\mathcal{O}}^{\top}$, $n \in (0,1]$, με $n \geq n'$. Έχει επίσης αποδειχθεί ότι ο αλγόριθμος πάντοτε τερματίζει σε πολυωνυμικό χρόνο ως προς το μέγεθος της οντολογίας εισόδου.

Κανονική Μορφή		**Κανόνας Συμπλήρωσης**
$\langle A_1 \sqcap ... \sqcap A_t \sqsubseteq B, n\rangle$	R1	**Αν** $\langle A_1, n_1\rangle \in S(X), ..., \langle A_l, n_l\rangle \in S(X)$, $\langle A_1 \sqcap ... \sqcap A_t \sqsubseteq B, k\rangle \in \mathcal{O}$ και $\langle B, m\rangle \notin S(X)$ όπου $m = min(n_1, ..., n_l, k)$ **τότε** $S(X) := S(X) \cup \{\langle B, m\rangle\}$.
$\langle A \sqsubseteq \exists r.B, n\rangle$	R2	**Αν** $\langle A, n\rangle \in S(X), \langle A \sqsubseteq \exists r.B, k\rangle \in \mathcal{O}$ και $\langle X, B, m\rangle \notin R(r)$ όπου $m = min(n, k)$ **τότε** $R(r) := R(r) \cup \{\langle X, B, m\rangle\}$.
$\langle \exists r.A \sqsubseteq B, n\rangle$	R3	**Αν** $\langle X, Y, n_1\rangle \in R(r), \langle A, n_2\rangle \in S(Y)$, $\langle \exists r.A \sqsubseteq B, n_3\rangle \in \mathcal{O}$ και $\langle B, m\rangle \notin S(X)$ όπου $m = min(n_1, n_2, n_3)$ **τότε** $S(X) := S(X) \cup \{\langle B, m\rangle\}$.
$r \sqsubseteq s$	R4	**Αν** $\langle X, Y, n\rangle \in R(r), r \sqsubseteq s \in \mathcal{O}$ και $\langle X, Y, n\rangle \notin R(s)$ **τότε** $R(s) := R(s) \cup \{\langle X, Y, n\rangle\}$.
$r \circ s \sqsubseteq t$	R5	**Αν** $\langle X, Y, n_1\rangle \in R(r), \langle Y, Z, n_2\rangle \in R(s)$, $r \circ s \sqsubseteq t \in \mathcal{O}$ και $\langle X, Z, m\rangle \notin R(t)$ όπου $m = min(n_1, n_2)$ **τότε** $R(t) := R(t) \cup \{\langle X, Z, m\rangle\}$.

***Πίνακας** 7: Αξιώματα κανονικής μορφής και κανόνες συμπλήρωσης για την f-EL⁺*

5.2.3 Βελτιωμένος Αλγόριθμος Υπολογισμού Αντιστοιχίσεων

Όπως έχει ήδη αναφερθεί, ένα από τα βασικότερα προβλήματα που συναντάται κατά την υλοποίηση των αλγορίθμων δομικής υπαγωγής για τις $\mathcal{EL}^+$ και $f - \mathcal{EL}^+$, οι οποίες περιεγράφησαν στην προηγούμενη ενότητα, είναι η ανάπτυξη ενός αποδοτικού τρόπου εύρεσης του επόμενου κανόνα συμπλήρωσης που πρέπει να εφαρμοστεί. Αν η αναζήτηση για τον επόμενο κανόνα γίνει με αλγόριθμο εξαντλητικής αναζήτησης – ωμής βίας, δεν μπορούμε να αναμένουμε ικανοποιητικό χρόνο εκτέλεσης για μεγάλες οντολογίες εισόδου. Ως λύση στο πρόβλημα αυτό προτείνεται στη βιβλιογραφία ([12]) η ακόλουθη βελτιωμένη έκδοση του αλγορίθμου υπολογισμού των αντιστοιχίσεων. Η βελτιωμένη αυτή έκδοση του αλγορίθμου χρησιμοποιεί ένα σύνολο ουρών, μία για κάθε ονοματική έννοια που συναντάται στην οντολογία εισόδου, προκειμένου να καθοδηγηθεί στην εφαρμογή των κανόνων συμπλήρωσης. Διαισθητικά, η ουρά περιέχει τις τροποποιήσεις που οφείλεται να γίνουν στα σύνολα $S(A)$ και $R(r)$. Οι πιθανές εγγραφές μίας ουράς είναι της μορφής

$$B_1 \sqcap ... \sqcap B_m \rightarrow \langle B', n'\rangle \text{ και } \langle \exists r.B, n\rangle,$$

όπου $B_1, ..., B_m$, B και B' είναι ονόματα εννοιών, r όνομα ρόλου, $m \geq 0$ και $n, n' \in (0, 1]$. Για την περίπτωση $m = 0$ η εγγραφή τύπου $B_1 \sqcap ... \sqcap B_m \rightarrow \langle B', n'\rangle$ της ουράς γράφεται απλά ως $\langle B', n'\rangle$. Επίσης:

1. μία εγγραφή τύπου $B_1 \sqcap ... \sqcap B_m \rightarrow \langle B', n' \rangle$ στην $queue(A)$ σημαίνει ότι η εγγραφή $\langle B', k \rangle$, όπου $k = min(n', n_1, ..., n_m)$ πρέπει να εισαχθεί στο $S(A)$, αν αυτό ήδη περιέχει καταχωρίσεις για τις έννοιες $B_1, ..., B_m$, δηλαδή αν περιέχει τις εγγραφές $\langle B_1, n_1 \rangle, ..., \langle B_m, n_m \rangle$, και
2. μία εγγραφή τύπου $\langle \exists r.B, n \rangle$ στην $queue(A)$ σημαίνει ότι η τριάδα $\langle A, B, n \rangle$ θα πρέπει να εισαχθεί στο $R(r)$.

Το γεγονός ότι μία τέτοια προσθήκη μπορεί να ενεργοποιήσει άλλους κανόνες πρέπει να ληφθεί υπ' όψιν επεκτείνοντας καταλλήλως τις ουρές όποτε λαμβάνει χώρα η προσθήκη. Όπως ακριβώς και στην περίπτωση της $\mathcal{EL}^+$, θεωρούμε μία αντιστοίχηση $\hat{\mathcal{O}}$ από ονόματα εννοιών σε σύνολα από εγγραφές ουρών, βάσει μίας κανονικοποιημένης οντολογίας εισόδου $\mathcal{O}$, ως εξής:

1. Αν $\langle A_1 \sqcap ... \sqcap A_m \sqsubseteq B, n \rangle \in \mathcal{O}$ και $A = A_i$, τότε

   $$A_1 \sqcap ... \sqcap A_{i-1} \sqcap A_{i+1} \sqcap ... \sqcap A_m \rightarrow \langle B, n \rangle \in \hat{\mathcal{O}}(A).$$

   Στην ειδική περίπτωση όπου $m = 1$ εισάγεται στην ουρά η εγγραφή $A_1 \rightarrow \langle B, n \rangle$, αντί της $\langle B, n \rangle$ που θα όριζε μία επέκταση του αντιστοίχου σταδίου του κλασικού σαφούς αλγορίθμου. Ο πλεονασμός αυτός εισάγεται για την εξασφάλιση της ορθότητας των τελικών αποτελεσμάτων, και συγκεκριμένα των βαθμών βεβαιότητας. Με τον τρόπο αυτό η διαδικασία *επεξεργασία* εισάγει το σωστό βαθμό βεβαιότητας κάθε φορά στα σύνολα S των υπαγουσών εννοιών, ενώ αν αυτή η τροποποίηση δεν είχε συμβεί αυτό δε γινόταν.

   Ένα παράδειγμα οντολογίας όπου γίνεται φανερή η επίδραση της τροποποίησης αυτής είναι η παρακάτω οντολογία:

   Ας υποθέσουμε εμφανίζονται τα ακόλουθα αξιώματα σε μία οντολογία $\mathcal{O}$:

   $$\langle CityCar \sqsubseteq Car, 0.8 \rangle \text{ και } \langle Car \sqsubseteq MeansOfTransport, 0.9 \rangle$$

   Τα αξιώματα αυτά δεν έχουν κάποιο ιδιαίτερο φυσικό νόημα, αλλά από αυτά μπορεί να διαπιστωθεί ότι ισχύει $\langle CityCar \sqsubseteq_{\mathcal{O}} MeansOfTransport, 0.8 \rangle$. Σε περίπτωση που δε γινόταν η παραπάνω τροποποίηση στον αλγόριθμο, τότε αυτός θα επέστρεφε τον λανθασμένο βαθμό βεβαιότητας 0.9 καθώς σε κάποια στιγμή της εκτέλεσης θα εισαγόταν στην ουρά του $CityCar$ η εγγραφή $\langle MeansOfTransport, 0.9 \rangle$. Με την τροποποίηση αυτή όμως καθοδηγείται η διαδικασία *επεξεργασία* να λάβει υπ' όψιν τον βαθμό 0.8 του πρώτου αξιώματος κατά τον υπολογισμό της αλυσίδας υπαγωγών. Αυτό έχει σαν αποτέλεσμα να υπολογίζεται σωστά ο τελικός βαθμός.

   Το πρόβλημα αυτό θα μπορούσε να μην επιλυθεί στο στάδιο αυτό, αλλά να ελέγχεται από τη διαδικασία *επεξεργασία* του θεωρητικού αλγορίθμου. Μία τέτοια αλλαγή όμως θα απαιτούσε έλεγχο κάθε υποψήφιας εγγραφής ουράς μία προς μία, γεγονός το οποίο θα εισήγαγε σημαντική καθυστέρηση στον αλγόριθμο συλλογιστικής. Ακόμα περισσότερο, οι αλλαγές αυτές θα διέφεραν από φορά σε φορά, οπότε δε θα ήταν ιδιαίτερο το κέρδος από την εφαρμογή λύσεων επαναχρησιμοποίησης αποτελεσμάτων, όπως οι κρυφές μνήμες. Εξ' αιτίας όλων των παραπάνω κρίνεται η λύση η οποία δόθηκε η βέλτιστη, καθώς διατηρεί το νόημα μίας εγγραφής ουράς αλλά, δεν απαιτεί πρακτικά περισσότερο χρόνο εκτέλεσης, δεν απαιτεί αλλαγές στη διαδικασία *επεξεργασία* και, τέλος, οι χωρικές απαιτήσεις του αλγορίθμου παραμένουν ουσιαστικά αμετάβλητες.
2. Αν $\langle A \sqsubseteq \exists r.B, n \rangle \in \mathcal{O}$, τότε $\langle \exists r.B, n \rangle \in \hat{\mathcal{O}}(A)$.

Αντιστοίχως, για κάθε έννοια $\exists r.A$ συμβολίζεται ως $\hat{\mathcal{O}}(\exists r.A)$ το ελάχιστο σύνολο εγγραφών ουράς τέτοιο ώστε αν $\langle \exists r.A \sqsubseteq B, n\rangle \in \mathcal{O}$, τότε $\langle B, n\rangle \in \hat{\mathcal{O}}(\exists r.A)$. Όπως γίνεται φανερό και στους αλγορίθμους 6 και 7, η εγγραφή στην ουρά δεν μπορεί να παραμείνει με τον μέγιστο αυτό βαθμό n, αν η αιτία που οδήγησε στην εισαγωγή της στην ουρά έχει βαθμό μικρότερο από n. Το γεγονός αυτό εισάγει μία σημαντική αύξηση στο πλήθος των ενεργειών που πρέπει να εκτελέσει ο αλγόριθμος προκειμένου να ταξινομήσει την οντολογία, αν και η θεωρητική πολυπλοκότητα δεν αλλάζει.

Στον βελτιωμένο αλγόριθμο η $queue(A)$ αρχικοποιείται με $\hat{\mathcal{O}}(A) \cup \hat{\mathcal{O}}(\top)$ – αφού το σύνολο $S(A)$ αρχικοποιείται με $\{\langle A, 1\rangle, \langle \top, 1\rangle\}$ –. Αυτό σημαίνει πως εισάγουμε στην $queue(A)$ τις άμεσες συνέπειες του να είναι το A ένα στιγμιότυπο του εαυτού του και του $\top$. Στη συνέχεια αφαιρούμε επαναληπτικά εγγραφές από τις ουρές και τις επεξεργαζόμαστε με χρήση της διαδικασίας *επεξεργασία* (Αλγόριθμος 6). Συγκεκριμένα, η διαδικασία *επεξεργασία*(A, X) καλείται όταν η ουρά της έννοιας A δεν ήταν κενή και εξήχθη από την $queue(A)$ το στοιχείο X προκειμένου να επεξεργαστεί. Η διαδικασία *επεξεργασία−νέας−ακμής*(A, r, B, n) (Αλγόριθμος 7) καλείται από την *επεξεργασία* προκειμένου να διαχειριστεί της συνέπειες της προσθήκης ενός νέου ζεύγους $\langle A, B, n\rangle$ στο $R(r)$.

Με το σύμβολο $\sqsubseteq^*_{\mathcal{O}}$ συμβολίζουμε το ανακλαστικό – μεταβατικό κλείσιμο των αξιωμάτων ιεραρχίας ρόλων της $\mathcal{O}$. Η επεξεργασία των ουρών συνεχίζει μέχρις ότου όλες οι ουρές είναι αδειάσουν, ενώ μπορεί να παρατηρηθεί ότι ο βελτιωμένος αυτός αλγόριθμος δεν απαιτείται να διενεργεί κανέναν έλεγχο περί του ποιου κανόνα συμπλήρωσης πρέπει να εφαρμόσει στη συνέχεια.

Διαδικασία επεξεργασία(A, X)

Αν $X = A_1 \sqcap ... \sqcap A_l \rightarrow \langle B, k\rangle$ **και** $\langle A_1, n_1\rangle \in S(A), ..., \langle A_l, n_l\rangle \in S(A)$ **τότε**

Αν $\langle B, f\rangle \notin S(A)$ **για οποιοδήποτε** $f \in (0, 1]$ **τότε**

$S(A) := S(A) \cup \{\langle B, m\rangle\}$**, όπου** $m = min(n_1, ..., n_l, k)$

$queue(A) := queue(A) \cup \hat{\mathcal{O}}(B)$

Για όλα τα ονόματα εννοιών A' **και ονόματα ρόλων** r **όπου**

$\langle A', A, n\rangle \in R(r)$

$queue(A') := queue(A') \cup \hat{\mathcal{O}}(\exists r.B)$**, όπου καμία εγγραφή δε θα έχει βαθμό μεγαλύτερο του** $min(m, n)$

αλλιώς αν $\langle B, f\rangle \in S(A)$ **και** $m > f$**, όπου** $m = min(n_1, ..., n_l, k)$

$S(A) := S(A) \cup \{\langle B, m\rangle\}$

Αν $X = \langle \exists r.B, n\rangle$ **και** $\langle A, B, n\rangle \notin R(r)$ **τότε**

επεξεργασία−νέας−ακμής(A, r, B, n)

Τέλος Διαδικασίας

Αλγόριθμος 6: *Αλγόριθμος Επεξεργασίας Ουρών: επεξεργασία(A, X)*

Διαδικασία επεξεργασία-νέας-ακμής(A, r, B, n)

Για όλα τα ονόματα ρόλων s **όπου** $r \sqsubseteq^*_{\mathcal{O}} s$

Αν $\langle A, B, m\rangle \notin R(s) \forall m \in (0,1]$ **ή** $\langle A, B, m\rangle \in R(s) \wedge n > m$ **τότε**

$R(s) := R(s) \cup \{\langle A, B, n\rangle\}$

$queue(A) := queue(A) \cup \bigcup_{\{B' | \langle B', m\rangle \in S(B)\}} \hat{\mathcal{O}}(\exists s.B')$, **όπου καμία εγγραφή δε θα έχει βαθμό μεγαλύτερο του** $min(m, n)$

Για όλα τα ονόματα εννοιών A' **και ονόματα ρόλων** t, u **όπου** $t \circ s \sqsubseteq u \in \mathcal{O}$ **και** $\langle A', A, k\rangle \in R(t)$ **και** $\langle A', B, m\rangle \notin R(u)$, **όπου** $m = min(n, k)$

επεξεργασία-νέας-ακμής(A', u, B, m)

Για όλα τα ονόματα εννοιών B' **και ονόματα ρόλων** t, u **όπου** $s \circ t \sqsubseteq u \in \mathcal{O}$ **και** $\langle B, B', k \in R(t)$ **και** $\langle A, B', m\rangle \notin R(u)$, **όπου** $m = min(n, k)$

επεξεργασία-νέας-ακμής(A, u, B', m)

Τέλος Διαδικασίας

Αλγόριθμος 7: *Αλγόριθμος Επεξεργασίας Ουρών: επεξεργασία-νέας-ακμής(A, r, B, n)*

Ο παραπάνω βελτιωμένος αλγόριθμος εκτελείται σε πολυωνυμικό χρόνο ($O(n^4)$) και είναι συνεπής και πλήρης, καθώς μετά τον τερματισμό ισχύει $\langle B, n\rangle \in S(A)$ αν και μόνο αν $\langle A \sqsubseteq_{\mathcal{O}} B, n\rangle$, για κάθε $A, B \in CN^{\top}_{\mathcal{O}}$ και $n \in (0, 1]$. Απόδειξη του παραπάνω μπορεί να γίνει με παρόμοιο τρόπο με αυτόν του [14] για την $\mathcal{EL}^+$.

5.2.4 Υπολογισμός Άμεσων Σχέσεων Υπαγωγής

Όπως ακριβώς και στην περίπτωση της $\mathcal{EL}^+$, το αποτέλεσμα της προηγούμενης διαδικασίας είναι τα υπολογισθέντα σύνολα S και R για κάθε όνομα έννοιας και ρόλου της οντολογίας, τα οποία περιέχουν και πληροφορίες βαθμού βεβαιότητας. Αυτή η μορφή των αποτελεσμάτων δεν είναι πάντα η επιθυμητή, και γι' αυτό υλοποιείται ένα επιπλέον στάδιο υπολογισμών, στο οποίο και υπολογίζονται οι άμεσες σχέσεις υπαγωγής.

Παραδείγματος χάριν, για μία έννοια A το σύνολο των εννοιών B που την υπάγουν άμεσα πληροί τις ακόλουθες ιδιότητες:

1. $\langle A \sqsubseteq_{\mathcal{O}} B, n\rangle$
2. $\nexists B' \notin \{A, B\} | \langle A \sqsubseteq_{\mathcal{O}} B', n_1\rangle \wedge \langle B' \sqsubseteq_{\mathcal{O}} B, n_2\rangle$

όπου $n, n_1, n_2 \in (0, 1]$, ενώ οι αντίστοιχες ιδιότητες που πληροί μία έννοια A η οποία υπάγει άμεσα ένα σύνολο εννοιών B είναι:

1. $\langle B \sqsubseteq_{\mathcal{O}} A, n\rangle$
2. $\nexists B' \notin \{A, B\} | \langle B \sqsubseteq_{\mathcal{O}} B', n_1\rangle \wedge \langle B' \sqsubseteq_{\mathcal{O}} A, n_2\rangle$

όπου και πάλι $n, n_1, n_2 \in (0, 1]$.

Τα σύνολα αυτά των εννοιών δημιουργούν έναν συνεκτικό ακυκλικό γράφο (Directed Acyclic Graph – DAG) υπαγωγής (subsumption DAG), ο οποίος αναπαριστά τις άμεσες σχέσεις μεταξύ εννοιών σε ολόκληρη την οντολογία.

Όπως έχει ήδη αναφερθεί, τα διάφορα σύνολα υπαγόντων και υπαγομένων δεν αναφέρονται μόνο σε συγκεκριμένες έννοιες, αλλά αφορούν ολόκληρες τις κλάσεις ισοδυναμίας στην οποία οι έννοιες αυτές ανήκουν. Τα πλεονεκτήματα της χρήσης ενός συνεκτικού ακυκλικού γράφου υπαγωγής σε σύγκριση με τα σύνολα υπαγουσών εννοιών (subsumer sets) είναι το γεγονός ότι ο γράφος αποτελεί πιο συνεπτυγμένη μορφή αναπαράστασης των δεδομένων (λιγότερος πλεονασμός κατά την αποθήκευση), ενώ επίσης υποστηρίζει πολύ εύκολα την περιδιάβαση (browsing) της ιεραρχίας υπαγωγών, καθώς από μία έννοια μπορούμε να επισκεφτούμε άμεσα τις υπαγόμενες απ' αυτήν και αυτές που την υπάγουν. Το βασικό μειονέκτημα είναι το ότι η εξέταση της αληθείας του ερωτήματος $\langle A \sqsubseteq^?_{\mathcal{O}} B, n\rangle$ απαιτεί την εξέταση της υπάρξεως μονοπατιού (reachability) στο γράφο από το B στο A, αντί ενός απλού ελέγχου στο σύνολο $S(A)$ των υπερκλάσεων της έννοιας A. Βεβαίως, αυτό το πρόβλημα παρακάμπτεται αν ο αλγόριθμος μπορεί να διατηρήσει και τις δύο δομές, δηλαδή τον γράφο και τα σύνολα S, στη μνήμη.

Ένας τρόπος υπολογισμού του συνεκτικού ακυκλικού γράφου υπαγωγής είναι μέσω του υπολογισμού για κάθε όνομα έννοιας A των ακολούθων συνόλων:

1. του συνόλου $SS(A) := \{\langle B, n_1\rangle \in S(A) | \langle A, n_2\rangle \notin S(B)\}$, το οποίο περιλαμβάνει τις έννοιες που υπάγουν αυστηρά την A, δηλαδή τις έννοιες που υπάγουν την A και δεν είναι ισοδύναμές της,
2. του συνόλου $DS(A) := SS(A) \setminus (\bigcup_{\langle B,n_1\rangle \in SS(A)} SS(B))$ των εννοιών που υπάγουν άμεσα την A και
3. του συνόλου $DS^-(A) := \{\langle B, n_1\rangle | \langle A, n_2\rangle \in DS(B)\}$ των άμεσα υπαγομένων από την Α εννοιών.

Τα σύνολα $DS(A)$ και $DS^-(A)$ αποτελούν μία αναπαράσταση του γράφου υπαγωγής. Παρ' όλα αυτά, ο τρόπος αυτός κατασκευής του γράφου υπαγωγής δεν είναι ο αποδοτικότερος: η κατασκευή του $DS^-(A)$ απαιτεί χρόνο $O(n^2)$, γι' αυτό και χρησιμοποιείται ένας διαφορετικός αλγόριθμος ([14]).

Η λειτουργία του αλγορίθμου είναι ίδια με τον αντίστοιχο της $\mathcal{EL}^+$. Για να εξηγηθεί η βασική ιδέα του αλγορίθμου που χρησιμοποιήθηκε, ας υποθέσουμε ότι έχουμε ήδη υπολογίσει ένα τμήμα του γράφου υπαγωγής για κάποιο υποσύνολο των ονομάτων εννοιών και είμαστε έτοιμοι να εισάγουμε την έννοια A στον γράφο. Ξεκινάμε υπολογίζοντας το σύνολο $SS(A)$ των εννοιών που υπάγουν αυστηρά την A. Τα στοιχεία του $S(A) \setminus SS(A)$ αποτελούν τις ισοδύναμες της A έννοιες. Για να βρούμε τις έννοιες που υπάγουν αυστηρά την A ανάμεσα στο $SS(A)$, προχωρούμε ως εξής:

1. Αν όλα τα στοιχεία του $SS(A)$ ανήκουν στον ήδη υπολογισθέντα γράφο, τότε μπορούμε να βρούμε τις έννοιες που υπάγουν άμεσα την A χρησιμοποιώντας ένα απλό αλγόριθμο διάσχισης γράφου για να "σημειώσουμε" όλες τις έννοιες του γράφου που υπάγουν αυστηρά στοιχεία του $SS(A)$. Οι έννοιες που αναζητούμε τότε είναι όλα τα στοιχεία του $SS(A)$ που δεν έχουν "σημειωθεί".

2. Αν υπάρχουν στοιχεία του $SS(A)$ τα οποία δεν ανήκουν στον μέχρι στιγμής υπολογισθέντα γράφο, τότε πρέπει πρώτα να εισάγουμε στο γράφο αυτά τα στοιχεία (με αναδρομικές κλήσεις της συνάρτησης εισαγωγής) προτού εισάγουμε την A. Με τον τρόπο αυτό είμαστε βέβαιοι ότι όταν εισαχθεί το όνομα μίας έννοιας στον γράφο τότε τη στιγμή εκείνοι όλες οι έννοιες οι οποίες την υπάγουν θα βρίσκονται ήδη στον γράφο, ενώ σ' αυτόν δε θα βρίσκεται καμία έννοια που υπάγεται από αυτήν. Συνεπώς, ο αλγόριθμος αυτός δε χρειάζεται να υπολογίζει τις άμεσα υπαγόμενες έννοιες, αλλά αρκεί να επεκτείνουμε το σύνολο των άμεσα υπαγόμενων από την έννοια B εισάγοντας την έννοια A, εάν βρεθεί ότι η Bυπάγει άμεσα την A.

Τα πλαίσια 8, 9 και 10 περιέχουν την αναπαράσταση του παραπάνω αλγορίθμου σε ψευδοκώδικα. Το σύνολο $\textit{γονείς}(A)$ χρησιμοποιείται για να αποθηκεύει τις έννοιες που υπάγουν άμεσα την A, το σύνολο $\textit{παιδιά}(A)$ περιέχει τις άμεσα υπαγόμενες από την A έννοιες, ενώ τέλος το σύνολο $\textit{ισοδύναμες}(A)$ περιέχει τις ισοδύναμες της A έννοιες. Η περιγραφή του αλγορίθμου χρειάζεται προσοχή, καθώς δεν φανερώνει την ήδη αναφερθείσα διαφορά ανάμεσα στο όνομα μίας έννοιας και σ' έναν κόμβο του γράφου: ο κόμβος του γράφου στην ουσία αποτελεί κλάση ισοδυναμία εννοιών. Αυτό έχει σαν αποτέλεσμα διάφορες λειτουργίες να μην υλοποιούνται απ' ευθείας, αλλά να πρέπει κάθε φορά να ενημερωθούν όλες οι ισοδύναμες έννοιες. Για παράδειγμα, αν εισάγουμε στο σύνολο $\textit{γονείς}(A)$ μία έννοια, τότε αυτή πρέπει να εισαχθεί και στα σύνολα $\textit{γονείς}$ όλων των ισοδυνάμων της A εννοιών.

Διαδικασία $\textit{υπολογισμός} - \textit{ΣΑΓ}()$

Για όλα τα ονόματα εννοιών $X \in CN_O^{\top}$

$\textit{ταξινομημένη}(X) := \textit{ψευδές}$

$\textit{γονείς}(X) := \textit{παιδιά}(X) := \textit{ισοδύναμες}(X) := \emptyset$

Για κάθε όνομα έννοιας $A \in CN_O^{\top}$

Αν δεν ισχύει ότι $\textit{ταξινομημένη}(A)$ **τότε**

$\textit{ΣΑΓ} - \textit{ταξινόμηση}(A)$

Τέλος Διαδικασίας

Αλγόριθμος 8: *Αλγόριθμος υπολογισμού ΣΑΓ υπαγωγής: υπολογισμός-ΣΑΓ()*

Διαδικασία $ΣΑΓ-ταξινόμηση(A)$

$υποψήφιοι := \emptyset$

Για κάθε $\langle B, n\rangle \in S(A)$

Αν $\langle A, m\rangle \in S(B)$ **τότε**

$ταξινομημένη(B) := αληθές$

$ισοδύναμες(A) := ισοδύναμες(A) \cup \{\langle B, m\rangle\}$

αλλιώς

Αν δεν ισχύει ότι $ταξινομημένη(B)$ **τότε**

$ΣΑΓ-ταξινόμηση(B)$

$υποψήφιοι := υποψήφιοι \cup \{\langle B, m\rangle\}$

$ΣΑΓ-εισαγωγή(A, υποψήφιοι)$

$ταξινομημένη(A) := αληθές$

Τέλος Διαδικασίας

Αλγόριθμος 9: *Αλγόριθμος υπολογισμού ΣΑΓ υπαγωγής: ΣΑΓ-ταξινόμηση(A)*

Διαδικασία $ΣΑΓ-εισαγωγή(A, υποψήφιοι)$

Για κάθε $X \in CN_{\mathcal{O}}^{\top}$

$σημειωμένο(X) := ψευδές$

Για κάθε $\langle B, n\rangle \in υποψήφιοι$

Για κάθε $\langle X, m\rangle \in γονείς(B)$

$σημειωμένο(X) := αληθές$

'Οσο υπάρχουν $X, Y \in CN_{\mathcal{O}}^{\top}$ **τέτοια ώστε** $σημειωμένο(X)$ **,** $\langle Y, n\rangle \in γονείς(X)$ **και δεν ισχύει ότι** $σημειωμένο(Y)$ **επανάλαβε**

$σημειωμένο(Y) := αληθές$

$γονείς(A) := γονείς(A) \cup \{\langle B, n\rangle \in υποψήφιοι : σημειωμένο(B) = ψευδές\}$

Για κάθε $\langle B, n\rangle \in γονείς(A)$

$παιδιά(B) := παιδιά(B) \cup \{\langle A, n\rangle\}$

Τέλος Διαδικασίας

Αλγόριθμος 10: *Αλγόριθμος υπολογισμού ΣΑΓ υπαγωγής: ΣΑΓ-εισαγωγή(A, υποψήφιοι)*

5.3 Σχόλια και Συγκρίσεις

Στην ενότητα αυτή θα συγκριθούν οι αλγόριθμοι που ακολουθούνται σε κάθε στάδιο της διαδικασίας συλλογιστικής για τις δύο Περιγραφικές Λογικές $\mathcal{EL}^+$ και $f-\mathcal{EL}^+$ και θα αναφερθούν ορισμένα σχόλια.

5.3.1 Μετατροπή σε Κανονική Μορφή

Οι δύο αλγόριθμοι δεν παρουσιάζουν ιδιαίτερες διαφορές πέραν των αναμενόμενων στα στάδια της κανονικοποίησης.

5.3.2 Αλγόριθμος Υπολογισμού Αντιστοιχίσεων

Συγκρίνοντας τους αλγορίθμους για τις δύο Περιγραφικές Λογικές στο στάδιο του υπολογισμού των αντιστοιχίσεων (mappings), μπορούμε να διαπιστώσουμε τη σημαντική ομοιότητά τους. Κάθε στάδιο του αλγορίθμου για την $f-\mathcal{EL}^+$ είναι όμοιο με το αντίστοιχο στάδιο για την $\mathcal{EL}^+$, με μόνη επέκταση την ανάγκη συμπερίληψης εντολών για την επεξεργασία των βαθμών βεβαιότητας των αξιωμάτων. Βέβαια, ο αλγόριθμος ασαφούς συλλογιστικής απέχει αρκετά από το να είναι μία απλή επέκταση του αντίστοιχου κλασικού αλγορίθμου. Παρατηρούνται τέσσερις βασικές διαφορές: Οι πρώτες δύο έχουν να κάνουν με τις δύο επιτρεπτές μορφές των στοιχείων ουράς, ενώ οι επόμενες σχετίζονται με αλλαγές στη διαδικασία του αλγορίθμου. Οι αλλαγές αυτές εξηγούνται στη συνέχεια:

1. Κατ' αρχάς, ο αλγόριθμος εύρεσης των απαιτούμενων εγγραφών ουράς για κάθε περιγραφή ονοματικής κλάσης παρουσιάζει μία μικρή διαφορά: δεν επιτρέπει την εισαγωγή εγγραφών της μορφής $\langle B, n\rangle$ στην ουρά. Αν πρέπει να εισαχθεί μία τέτοια εγγραφή στην ουρά, στην πραγματικότητα θα εισαχθεί μία εγγραφή του τύπου $A \rightarrow \langle B, n\rangle$, όπου A η έννοια – περιγραφή για την οποία υπολογίζεται το σύνολο απαιτουμένων εγγραφών ουρών. Η κίνηση αυτή εκ πρώτης όψεως δείχνει να αποτελεί πλεονασμό σε σχέση με το αντίστοιχο στάδιο του κλασικού (σαφούς) αλγορίθμου, είναι όμως μία απαραίτητη ενέργεια καθώς διασφαλίζει τη σωστή εφαρμογή του κανόνα συμπλήρωσης R1 (βλ. Πίνακα 7, σελ. 36) από τη διαδικασία $\textit{επεξεργασία}(A, X)$ του αλγορίθμου με το μικρότερο δυνατό χρονικό κόστος, ενώ το χωρικό κόστος παραμένει πρακτικά αμετάβλητο.
2. Η δεύτερη διαφορά εντοπίζεται στις ενέργειες που εκτελεί ο βελτιωμένος αλγόριθμος προκειμένου να επιτύχει την ορθή εφαρμογή του κανόνα συμπλήρωσης R3 (βλ. Πίνακα 7, σελ. 36). Οι διαφορές αυτές μπορούν να διαπιστωθούν στα σημεία όπου οι δύο αλγόριθμοι απαιτούν την εισαγωγή απαιτούμενων εγγραφών ουρών για περιγραφές τύπου $\exists r.A$, δηλαδή για σύνολα απαιτούμενων εγγραφών ουρών μορφής $\hat{\mathcal{O}}(\exists r.A)$. Η διαφορά έγκειται στο γεγονός ότι ο βαθμός βεβαιότητας κάθε εγγραφής ουράς δε θα πρέπει να υπερβαίνει το βαθμό βεβαιότητας ισχύος του αιτίου που προκαλεί την εισαγωγή του. Η διασφάλιση αυτού του κανόνα απαιτεί τον έλεγχο όλων των στοιχείων του συνόλου απαιτούμενων εγγραφών και τη διόρθωση όσων βαθμών βεβαιότητας κρίνεται αναγκαίο. Η αλλαγή αυτή σε σύγκριση με τον αλγόριθμο σαφούς συλλογιστικής αυξάνει το πλήθος των ενεργειών που πρέπει να εκτελεί ο αλγόριθμος κατά την ταξινόμηση μίας οντολογίας, γι' αυτό και έχουν γίνει ορισμένες βελτιστοποιήσεις σε επίπεδο υλοποίησης ώστε να περιοριστεί η επίδραση αυτών των ελέγχων.

3. Κατά τρίτον , αλλαγή παρατηρείται και στη διαδικασία *επεξεργασία*(A, X) : ο αλγόριθμος παρουσιάζει διαφορετική συμπεριφορά αν ένα ζεύγος $\langle B, f\rangle \notin S(A)$ για οποιοδήποτε $f \in (0, 1]$, απ' ότι αν υπάρχει μεν εγγραφή για την κλάση B, είναι δε διαφορετικού βαθμού απ' αυτόν που απαιτείται από τον κανόνα συμπλήρωσης R1, δηλαδή τον m. Στο σημείο αυτό μπορεί να παρατηρηθεί και άλλη μία απόκλιση: εφ' όσον στο στάδιο **αλλιώς** γνωρίζουμε ήδη πως υπάρχει εγγραφή για τη B στο $S(A)$, εξετάζουμε μόνο αν η εγγραφή έχει βαθμό βεβαιότητας μικρότερο ή ίσο με τον επιθυμητό (m) και αν ναι, τον ενημερώνουμε, δηλαδή τον αντικαθιστούμε με m. Αυτή η διαφοροποίηση δεν οδηγεί σε λανθασμένα αποτελέσματα, αφού λόγω της σημασιολογίας της $f - \mathcal{EL}^+$, ο βαθμός έχει την έννοια του ελαχίστου ορίου. Αυτό έχει ως συνέπεια η μόνη επιτρεπτή ανανέωση ενός βαθμού σε μία εγγραφή συνόλου αντιστοιχίσεων να είναι προς την κατεύθυνση της αύξησής του.

   Η ποιοτική αλλά και δομική αυτή διαφοροποίηση του αλγορίθμου δεν εμφανίζεται την περίπτωση της κλασικής $\mathcal{EL}^+$, καθώς στην περίπτωσή της δεν έχουμε βαθμούς βεβαιότητας και επομένως μία κλάση B μπορεί να προστεθεί μόνο μία φορά κατά την εκτέλεση του αλγορίθμου συλλογιστικής σε ένα σύνολο $S(A)$. Αυτό δε συμβαίνει στον ασαφή αλγόριθμο, καθώς ναι μεν μπορεί να έχει προστεθεί η κλάση B στο σύνολο $S(A)$ με κάποιον βαθμό n, αλλά μπορεί να χρειαστεί αυτός ο βαθμός να επικαιροποιηθεί εξ' αιτίας άλλων αξιωμάτων της οντολογίας με τον βαθμό m. Αν κάθε φορά εκτελείτο το σύνολο των ενεργειών – δηλαδή η εισαγωγή στοιχείων στις ουρές σαν δηλαδή να εισάγετο το $\langle B, m\rangle$ για πρώτη φορά στο $S(A)$ –, τότε ο αλγόριθμος δε θα τερμάτιζε.

4. Τέλος, αλλαγή παρατηρείται στη διαδικασία *επεξεργασία−νέας−ακμής*(A, r, B, n) , όπου η προσθήκη του $\langle A, B, n\rangle$ στο $R(s)$ λαμβάνει χώρα μόνο αν $\langle A, B, m\rangle \notin R(s) \forall m \in (0, 1]$ ή αν $\exists m \in (0, 1] | \langle A, B, m\rangle \in R(s)$ και $m < n$. Και σε αυτή την περίπτωση όσες νέες εισαγωγές γίνονται στο Σύνολο Ρόλων γίνονται προς την κατεύθυνση της αύξησης του βαθμού βεβαιότητας.

5.3.3 Υπολογισμός Άμεσων Σχέσεων Υπαγωγής

Ο τρόπος υπολογισμού που ακολουθείται για την $f - \mathcal{EL}^+$ είναι όμοιος με αυτόν που έχει αναφερθεί ήδη για την $\mathcal{EL}^+$. Συγκεκριμένα, οι βαθμοί βεβαιότητας δεν επηρεάζουν ιδιαίτερα τη ροή του αλγορίθμου, αλλά απλά λαμβάνονται υπ' όψιν προκειμένου να γίνει ορθή καταγραφή στα τελικά σύνολα που δημιουργούνται. Για να γίνει αυτό το σημείο πιο κατανοητό, πρέπει να αναρωτηθούμε αν, δεδομένου ότι ισχύει $\langle B, n\rangle \in S(A)$ για κάποια $A, B \in CN_{\mathcal{O}}^{\top}$ και ότι $n \in (0, 1]$, επηρεάζεται ιδιαίτερα η πορεία του αλγορίθμου από το ποιο συγκεκριμένα είναι το n και, ακόμα περισσότερα, αν η πορεία που θα ακολουθηθεί και τα τελικά σύνολα στα οποία θα υπάρξει εγγραφή για το B θα ήταν διαφορετικά από αυτά της ίδιας οντολογίας σε Περιγραφική Λογική $\mathcal{EL}^+$, αγνοώντας δηλαδή τους βαθμούς ισχύος των ασαφών αξιωμάτων. Η απάντηση στο ερώτημα αυτό είναι ότι οι βαθμοί βεβαιότητας δεν επηρεάζουν ουσιαστικά τον αλγόριθμο, οπότε αρκεί να υλοποιηθεί αλγόριθμος αντίστοιχος του αλγορίθμου σαφούς λογικής που έχει ήδη περιγραφεί. Ο νέος αυτός αλγόριθμος πρέπει να λαμβάνει υπ' όψιν τους βαθμούς βεβαιότητας, αλλά απλά για την εύρεση του βαθμού με τον οποίο θα εισαχθεί μία κλάση σε ένα σύνολο και όχι για το ποιο θα είναι το σύνολο αυτό.

6. Περιγραφή Σχεδίασης Συστήματος

Στο κεφάλαιο αυτό θα αναλυθεί η γενική αρχιτεκτονική του συστήματος που υλοποιήθηκε, χωρίς αναφορά σε συγκεκριμένα σημεία του κώδικα όπως κλάσεις ή μεθόδους. Επίσης, αναλύονται οι απαιτήσεις του και οι δυνατότητες που αυτό παρέχει.

Δύο είναι τα κύρια τμήματα του συστήματος:

1. Πρώτον, ο πυρήνας της μηχανής συλλογιστικής. Ο πυρήνας είναι το κύριο τμήμα της μηχανής, το οποίο δέχεται μία οντολογία και, αναλόγως των επιθυμιών του χρήστη – πελάτη, ελέγχει την ισχύ ενός ερωτήματος ή εκτελεί άλλες ενέργειες. Το τμήμα αυτό είναι το πολυπλοκότερο και ουσιαστικότερο τμήμα του υλοποιηθέντος συστήματος.
2. Δεύτερον, το τμήμα της διεπαφής και της επικοινωνίας με το χρήστη. Το τμήμα αυτό, το οποίο λειτουργεί ως υπηρεσία (service) σε κάποιον υπολογιστή, καλεί το πρώτο τμήμα με τον κατάλληλο τρόπο για κάθε εισερχόμενο αίτημα συλλογιστικής, παραλαμβάνει το αποτέλεσμα και το επιστρέφει στον αιτούντα. Με τον όρο διεπαφή νοείται απλά μία διαπροσωπεία που δίνει τη δυνατότητα της παροχής υπηρεσιών συλλογιστικής για τις υποστηριζόμενες Περιγραφικές Λογικές από ένα σύστημα – εξυπηρετητή (server) σε κάποιον τοπικό ή απομακρυσμένο πελάτη, είτε αυτός είναι άνθρωπος, είτε είναι κάποιο πρόγραμμα.

Σχήμα 1: *Διάγραμμα Αρχιτεκτονικής Συστήματος Συλλογιστικής. Απεικονίζεται και ο χρήστης.*

6.1 Μηχανή Συλλογιστικής

Στο τμήμα αυτό θα αναφερθούν οι απαιτήσεις και οι δυνατότητες της μηχανής συλλογιστικής.

6.1.1 Απαιτήσεις Δεδομένων Εισόδου

Η μηχανή συλλογιστικής που υλοποιήθηκε υποστηρίζει τις Περιγραφικές Λογικές $\mathcal{EL}^+$ και $f-\mathcal{EL}^+$, όπως αυτές ορίζονται στα [13] και [12] αντίστοιχα. Η οντολογία μπορεί να βρίσκεται αποθηκευμένη είτε στο προσβάσιμο από τον τοπικό υπολογιστή σύστημα αρχείων, είτε ως πόρος προσβάσιμος μέσω HTTP. Το όνομα του αρχείου της οντολογίας εισόδου πρέπει να έχει κατάληξη ".owl", ενώ το συντακτικό του πρέπει να χρησιμοποιεί κάποια από τις υποστηριζόμενες από το OWL API μορφές:

1. OWL/XML

2. OWL/RDF
3. OWL Functional Syntax
4. Turtle (Terse RDF Triple Language)
5. KRSS και
6. OBO Flat File Format

Για όσες λειτουργίες υποστηρίζεται η εξαγωγή οντολογιών, αυτή γίνεται στην προκαθορισμένη μορφή OWL/XML.

Απαραίτητη προϋπόθεση προκειμένου να μπορέσει η μηχανή συλλογιστικής να λειτουργήσει και να ταξινομήσει την οντολογία είναι, πέραν των τετριμμένων (ύπαρξη αρχείων, σωστή σύνταξή τους κ.α.), αυτή να έχει την εκφραστικότητα της $\mathcal{EL}^+$. Η ίδια απαίτηση ισχύει προφανώς και στην περίπτωση που επιθυμείται η ταξινόμηση μίας $fuzzy - \mathcal{EL}^+$ οντολογίας, καθώς η μόνη διαφορά μεταξύ των δύο Περιγραφικών Λογικών είναι η υποστήριξη βαθμών ασάφειας στα αξιώματα υπαγωγής από τη δεύτερη. Ο τρόπος καταγραφής των βαθμών βεβαιότητας δεν επηρεάζει την εκφραστικότητα της οντολογίας, όπως θα δειχθεί στη συνέχεια.

Κατά το σχεδιασμό της υλοποίησης του συστήματος και προκειμένου αυτό να έχει τη μέγιστη συμβολή και την αμεσότερη ολοκλήρωση με το σύστημα FReS (Fuzzy Reasoning Services), το οποίο και υλοποιείται από την ομάδα Τεχνολογιών Γνώσης, δόθηκε ιδιαίτερη προσοχή στον ορισμό ενός τρόπου καταγραφής στην οντολογία του ασαφούς βαθμού ισχύος ενός αξιώματος. Έπρεπε να καθοριστεί η χρήση μίας σύνταξης ενιαίας ανάμεσα σε όλες τις διαφορετικές μηχανές συλλογιστικής που χρησιμοποιεί το σύστημα και να εξασφαλιστεί η απόλυτη υπακοή σε αυτήν, πράγμα σημαντικό για την διαλειτουργικότητα. Λόγω των παραπάνω και σε μία προσπάθεια να βρεθεί η μία κοινή τομή μεταξύ των απαιτήσεων όλων των απαιτήσεων και των λειτουργιών του συστήματος FReS, αποφασίστηκε να ακολουθηθεί το παρακάτω πρότυπο για τη μηχανή συλλογιστικής $f - \mathcal{EL}^+$:

1. Οι βαθμοί βεβαιότητας θα αναπαρίστανται ως αξιώματα επισημείωσης (annotations) αξιώματος της OWL 2. Το αντικείμενο το οποίο θα τα αναπαριστά στο OWL API θα είναι το `org.semanticweb.owl.model.Interface OWLAxiomAnnotationAxiom`.

   Γνωρίζουμε επίσης πως για κάθε επισημείωση στην OWL 2 μας ενδιαφέρουν δύο χαρακτηριστικά: τον Καθολικό Προσδιοριστή Πόρου (Universal Resource Indentifier, URI) της επισημείωσης και ένα πεδίο τιμής, το οποίο μπορεί να περιέχει έναν αριθμό, μία συμβολοσειρά (string) ή κάποιο άλλο στοιχείο. Επομένως:

2. Καταχωρίσεις βαθμών βεβαιότητας θα διαθέτουν URI το οποίο θα προκύπτει από τη συνένωση του URI της οντολογίας και του `#hasFuzziness`.

3. Το πεδίο τιμής κάθε επισημείωσης θα πρέπει να περιέχει μία συμβολοσειρά (string), η οποία θα αναπαριστά στην ίδια γραμμή δύο αριθμούς. Οι αριθμοί αυτοί θα διαχωρίζονται με έναν χαρακτήρα κενού και εάν είναι δεκαδικοί, θα χρησιμοποιούν ως χαρακτήρα υποδιαστολής την τελεία (.). Οι δύο αυτοί αριθμοί θα ορίζουν τα σημεία έναρξης και λήξης του κλειστού διαστήματος εντός του οποίου βρίσκεται ο βαθμός βεβαιότητας.

   Με βάση τη σημασιολογία της $f - \mathcal{EL}^+$ έχει νόημα μόνο ο πρώτος εκ των δύο αριθμών, ο οποίος και πρέπει να ανήκει στο διάστημα $[0, 1]$. Ο δεύτερος των αριθμών, ο οποίος προσδιορίζει το άνω όριο του διαστήματος, πρέπει να είναι ίσος με τη μονάδα.

Μία οντολογία θεωρείται οντολογία $f - \mathcal{EL}^+$ και συνεπώς ενεργοποιείται ο κατάλληλος αλγόριθμος για την επεξεργασία της όταν διαθέτει την εκφραστικότητα της $\mathcal{EL}^+$ και περιέχει τουλάχιστον μία τέτοια έγκυρη δήλωση βαθμού βεβαιότητας. Όσα από τα Γενικευμένα Αξιώματα Υπαγωγής (GCIs) μίας ασαφούς οντολογίας δε σχετίζονται με κάποιο αξίωμα επισημείωσης θεωρείται σιωπηρά πως ισχύουν με απόλυτη βεβαιότητα, είναι δηλαδή ο βαθμός βεβαιότητας ισχύος τους ίσος με τη μονάδα. Αν σε μία κατά τα άλλα σαφή οντολογία διαπιστωθεί η ύπαρξη ενός συντακτικά μη έγκυρου αξιώματος επισημείωσης, τότε αυτό αγνοείται και για την επεξεργασία της οντολογίας ακολουθείται ο αλγόριθμος για τη κλασική $\mathcal{EL}^+$.

Ως παράδειγμα συντακτικά ορθού ορισμού ενός αξιώματος υπαγωγής και του βαθμού βεβαιότητάς του σε μία οντολογία καταγεγραμμένη σε μορφή OWL/XML παρουσιάζεται η ακόλουθη οντολογία, η οποία απλά ορίζει ότι $\langle B \sqsubseteq A, 0.9 \rangle$:

```
<?xml version="1.0"?>

<!DOCTYPE Ontology [
   <!ENTITY owl "http://www.w3.org/2002/07/owl#" >
   <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#" >
   <!ENTITY owl2xml "http://www.w3.org/2006/12/owl2-xml#" >
   <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#" >
   <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#" >
   <!ENTITY sampleOntology
"http://www.semanticweb.org/ontologies/2009/6/sampleOntology.owl#" >
]>


<Ontology xmlns="http://www.w3.org/2006/12/owl2-xml#"
    xml:base="http://www.w3.org/2006/12/owl2-xml#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:owl2xml="http://www.w3.org/2006/12/owl2-xml#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:sampleOntology="http://www.semanticweb.org/ontologies/2009/6/sampleOnto
logy.owl#"
    URI="http://www.semanticweb.org/ontologies/2009/6/sampleOntology.owl">
   <Declaration>
      <Class URI="&sampleOntology;A"/>
   </Declaration>
   <SubClassOf>
      <Annotation annotationURI="&sampleOntology;hasFuzziness">
         <Constant datatypeURI="&xsd;string">0.9 1.0</Constant>
      </Annotation>
      <Class URI="&sampleOntology;B"/>
      <Class URI="&sampleOntology;A"/>
   </SubClassOf>
   <Declaration>
      <Class URI="&sampleOntology;B"/>
   </Declaration>
</Ontology>
```

6.1.2 Αρχιτεκτονική Συστήματος Συλλογιστικής

Όπως έχει ήδη αναφερθεί και στο κεφάλαιο "Σφάλμα: Δεν βρέθηκε η πηγή παραπομπής", η βασική υπηρεσία συλλογιστικής που προσφέρεται για τις Περιγραφικές Λογικές $\mathcal{EL}^+$ και $f-\mathcal{EL}^+$ είναι η **υπηρεσία ταξινόμησης (classification)** μία οντολογίας. Παρά τις πολλές διαφορετικές μεθόδους εκτέλεσης ερωτημάτων και εμφάνισης αποτελεσμάτων που προσφέρονται, η υπηρεσία ταξινόμησης είναι και η μόνη η οποία παρέχεται από την υλοποιηθείσα μηχανή συλλογιστικής.

Το σύστημα συλλογιστικής που υλοποιήθηκε δέχεται ως είσοδο ένα αρχείο οντολογίας, οι λεπτομέρειες σύνταξης και μορφής του οποίου θα αναλυθούν στην επόμενη ενότητα. Από το αρχείο αυτό ανακτώνται τα αξιώματα και οι επισημάνσεις βαθμών βεβαιότητας της οντολογίας, λαμβάνει χώρα η επεξεργασία τους και στη συνέχεια τα αποτελέσματα εξάγονται σε κάποια χρήσιμη για τον χρήστη μορφή.

Σημείωση: Στην ενότητα αυτή με την έννοια "χρήστης του συστήματος" δε νοείται αποκλειστικά κάποιος άνθρωπος – χρήστης. Μπορεί να εννοείται και κάποιο πρόγραμμα ή υπηρεσία – πελάτης το οποίο χρησιμοποιεί τη μηχανή είτε με την τοπική είτε με την απομακρυσμένη διεπαφή.

Ο καθορισμός του αρχείου εισόδου ή του ορισμού των ερωτημάτων και των λειτουργιών που καλείται να εκτελέσει η μηχανή συλλογιστικής υλοποιούνται από το σύστημα διεπαφής με τους χρήστες, το οποίο αναλύεται στη συνέχεια.

Οι λειτουργίες που λαμβάνουν χώρα σε ένα τυπικό σχέδιο χρήσης του συστήματος είναι οι εξής:

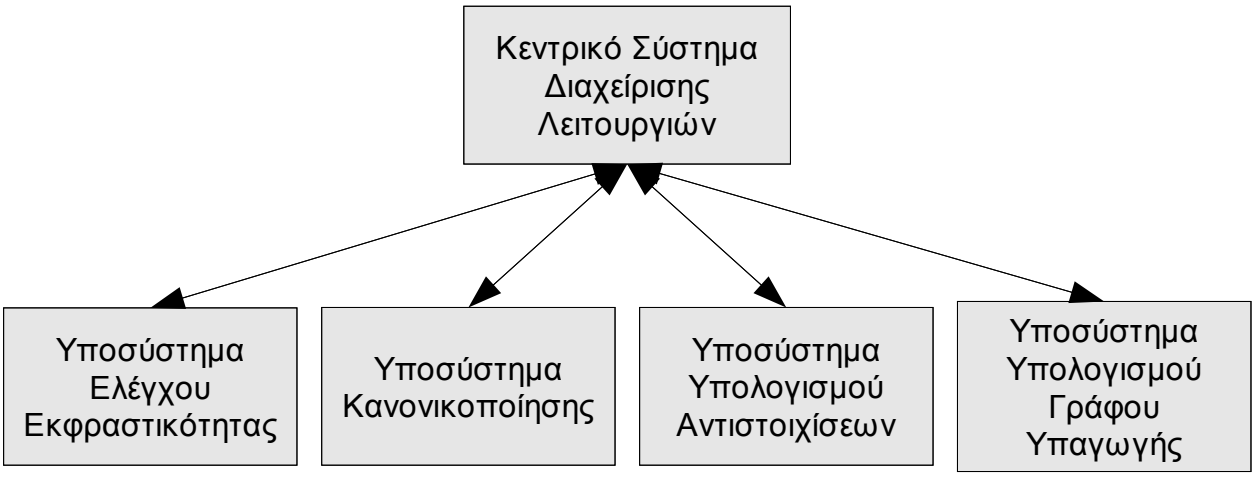
1. **Αρχικοποίηση και εκκίνηση** του κεντρικού συστήματος διαχείρισης λειτουργιών βάσει των επιθυμιών του χρήστη, όπως αυτές αναγνωρίζονται από το σύστημα διεπαφής.
2. **Εκτέλεση ελέγχου εκφραστικότητας** της οντολογίας: Στο στάδιο αυτό ελέγχεται αν η εισαχθείσα προς ταξινόμηση οντολογία είναι πράγματι οντολογία Περιγραφικής Λογικής $\mathcal{EL}^+$ ή $f-\mathcal{EL}^+$.
3. **Εκτέλεση προεπεξεργασίας** της οντολογίας: Στο στάδιο αυτό γίνονται ορισμένες μετατροπές αξιωμάτων της οντολογίας σε ισοδύναμα συγκεκριμένου τύπου, ώστε να καταστεί απλούστερος ο αλγόριθμος που υλοποιεί τα επόμενα στάδια της επεξεργασίας της οντολογίας. Το στάδιο αυτό δεν πρέπει να συγχέεται με το επόμενο στάδιο, αυτό της κανονικοποίησης των αξιωμάτων.
4. **Κανονικοποίηση των αξιωμάτων** της οντολογίας: Το υποσύστημα αυτό αναλαμβάνει τη μετατροπή όλων των αξιωμάτων σε συγκεκριμένες κανονικές μορφές, οι οποίες παρουσιάζονται στο κεφάλαιο "Σφάλμα: Δεν βρέθηκε η πηγή παραπομπής". Η επεξεργασία αυτή απαιτείται από τον θεωρητικό αλγόριθμο προκειμένου να διευκολυνθεί ο χειρισμός των αξιωμάτων από τα επόμενα στάδια
5. **Υπολογισμός των Αντιστοιχίσεων (mappings):** Το υποσύστημα υπολογισμού των αντιστοιχίσεων υλοποιεί την ομώνυμη λειτουργία, η οποία παρουσιάζεται στο κεφάλαιο της περιγραφής του αλγορίθμου. Γίνεται χρήση του βελτιωμένου αλγορίθμου, όπως αυτός παρουσιάζεται στη βιβλιογραφία ([13], [12]).
6. **Υπολογισμός του συνεκτικού ακυκλικού γράφου υπαγωγής (subsumption DAG)**: Το υποσύστημα αυτό αναλαμβάνει τον υπολογισμό του γράφου που απεικονίζει τις άμεσες σχέσεις υπαγωγής και τις σχέσεις ισοδυναμίας μεταξύ των εννοιών. Για τον υπολογισμό απαιτούνται οι πληροφορίες των συνόλων του προηγουμένου σταδίου. Η εκτέλεση του

σταδίου αυτού δεν είναι απαραίτητη για όλες τις περιπτώσεις λειτουργίας της μηχανής συλλογιστικής. Αν δηλαδή οι πληροφορίες που ζητούνται μπορούν να παρασχεθούν από τα σύνολα αντιστοιχίσεων, τότε δε λαμβάνει χώρα ο υπολογισμός του γράφου υπαγωγής.

7. **Μεταφορά του ελέγχου στο κεντρικό σύστημα διαχείρισης των λειτουργιών**, το οποίο απαντά τα ερωτήματα που τίθενται και εξάγει τα αποτελέσματα στις επιθυμητές από το χρήστη μορφές.

Κάθε μία από τις ανωτέρω αναφερόμενες λειτουργίες του συστήματος συλλογιστικής υλοποιείται από ένα ανεξάρτητο υποσύστημα. Κάθε υποσύστημα επικοινωνεί εν γένει μόνο με το κεντρικό σύστημα διαχείρισης των λειτουργιών της μηχανής: το κεντρικό σύστημα τροφοδοτεί με τα δεδομένα εισόδου κάθε υποσύστημα και δέχεται τα αποτελέσματά του, προκειμένου να δημιουργήσει τα δεδομένα εισόδου του επόμενου υποσυστήματος. Μετά το τέλος της λειτουργίας ενός υποσυστήματος ο έλεγχος δεν επανέρχεται καμία στιγμή σε αυτό, παρά μόνο αν απαιτείται να γίνει ανάγνωση των αποτελεσμάτων των υπολογισμών του.

Η προαναφερθείσα σχεδίαση του συστήματος, η οποία και απεικονίζεται στο Σχήμα 2, επελέγη με σκοπό τη διευκόλυνση της ανάπτυξης, της βελτίωσης και της αποσφαλμάτωσης του κώδικα.



Σχήμα 2: *Διάγραμμα Αρχιτεκτονικής Πυρήνα Συστήματος Συλλογιστικής.*

Στα επόμενα θα αναφερθούν με περισσότερη λεπτομέρεια η αρχιτεκτονική σχεδίαση και οι λειτουργίες του κάθε υποσυστήματος ξεχωριστά, ενώ θα παρουσιαστούν και σχετικά διαγράμματα ροής.

6.1.2.1 Κεντρικό Σύστημα Διαχείρισης Λειτουργιών

Βασική λειτουργία του συστήματος διαχείρισης είναι η επικοινωνία με το τμήμα της διεπαφής με το χρήστη και η επιτυχής εκτέλεση των ενεργειών που αυτός ορίζει. Το σύστημα διαχείρισης λειτουργιών καλείται με συγκεκριμένες παραμέτρους, φορτώνει την οντολογία στη μνήμη και καλεί διαδοχικά τα υπόλοιπα υποσυστήματα, τα οποία με τη σειρά τους υλοποιούν τον αλγόριθμο συλλογιστικής που έχει περιγραφεί. Στο τέλος της διαδικασίας αυτής, το κεντρικό σύστημα διαχείρισης είναι υπεύθυνο για την απάντηση των ερωτημάτων που πιθανώς έχουν τεθεί από τον χρήστη και την εξαγωγή των απαντήσεων με τον τρόπο που αυτός επιθυμεί. Για την

υλοποίηση πολλών από αυτές τις λειτουργίες, όπως η ανάκτηση, η επεξεργασία και η αποθήκευση – εξαγωγή οντολογιών, πολύ σημαντική ήταν η συμβολή του OWL API.

Συγκεκριμένα, το κεντρικό υποσύστημα διαχείρισης εκτελεί τις ακόλουθες λειτουργίες στον κύκλο λειτουργίας του:

1. **Ανάκτηση της οντολογίας** προς ταξινόμηση από το τοπικό σύστημα αρχείων ή μέσω του πρωτοκόλλου HTTP.
2. **Έλεγχος εκφραστικότητάς** της.
3. Εκκίνηση **διαδικασίας προεπεξεργασίας** της.
4. **Εξαγωγή της κανονικοποιημένης οντολογίας**, αν αυτό ζητείται.
5. Εκκίνηση **διαδικασίας υπολογισμού συνόλων αντιστοιχίσεων (mappings)**.
6. Εκκίνηση **διαδικασίας υπολογισμού του Συνεκτικού Ακυκλικού Γράφου Υπαγωγής (subsumption DAG)**, αλλά μόνο αν κάτι τέτοιο απαιτείται για την εκτέλεση των αιτημάτων του χρήστη – πελάτη.
7. **Διερεύνηση και απάντηση των ερωτημάτων του χρήστη** καθώς και εκτέλεση των λοιπών αιτημάτων αυτού.

Στο σημείο αυτό πρέπει να τονισθεί πως δεν είναι όλες οι λειτουργίες εξαγωγής στοιχείων και απάντησης ερωτημάτων διαθέσιμες και με τους δύο τρόπους διεπαφής με τη μηχανή συλλογιστικής (τοπικά ή μέσω Socket). Οι δυνατότητες που υποστηρίζονται μέσω της διεπαφής με Socket είναι ένα υποσύνολο των δυνατών λειτουργιών του συστήματος, όπως θα αναφερθεί στη συνέχεια.

Οι λειτουργίες 2 έως 6 αναλύονται σε ξεχωριστά υποσυστήματα, ενώ οι λειτουργίες 1 και 7 είναι τμήμα του κεντρικού συστήματος ελέγχου και υλοποιούνται με κλήσεις μεθόδων των υπολοίπων υποσυστημάτων.

Είναι αναμενόμενο πως σε οποιοδήποτε στάδιο αυτών των λειτουργιών ενδέχεται να προκύψουν σφάλματα, τα οποία είτε να είναι ήσσονος σημασίας – πράγμα το οποίο σημαίνει πως η διαδικασία συλλογιστικής μπορεί να συνεχιστεί –, είτε να είναι σφάλματα τα οποία επιβάλλουν τον τερματισμό της εκτέλεσης του προγράμματος. Τέτοια σφάλματα είναι, μεταξύ άλλων, η αποτυχία ανάκτησης της οντολογίας, η ανακάλυψη ότι αυτή έχει κάποια μη υποστηριζόμενη εκφραστικότητα ή κάποιο εσωτερικό σφάλμα κατά τη διαδικασία ταξινόμησης (classification). Σε κάθε περίπτωση εμφανίζονται τα κατάλληλα διευκρινιστικά μηνύματα στον χρήστη και εκτελούνται όλες οι απαραίτητες ενέργειες για τον ασφαλή τερματισμό της λειτουργίας του συστήματος.

6.1.2.2 Υποσύστημα Ελέγχου Εκφραστικότητας Οντολογίας

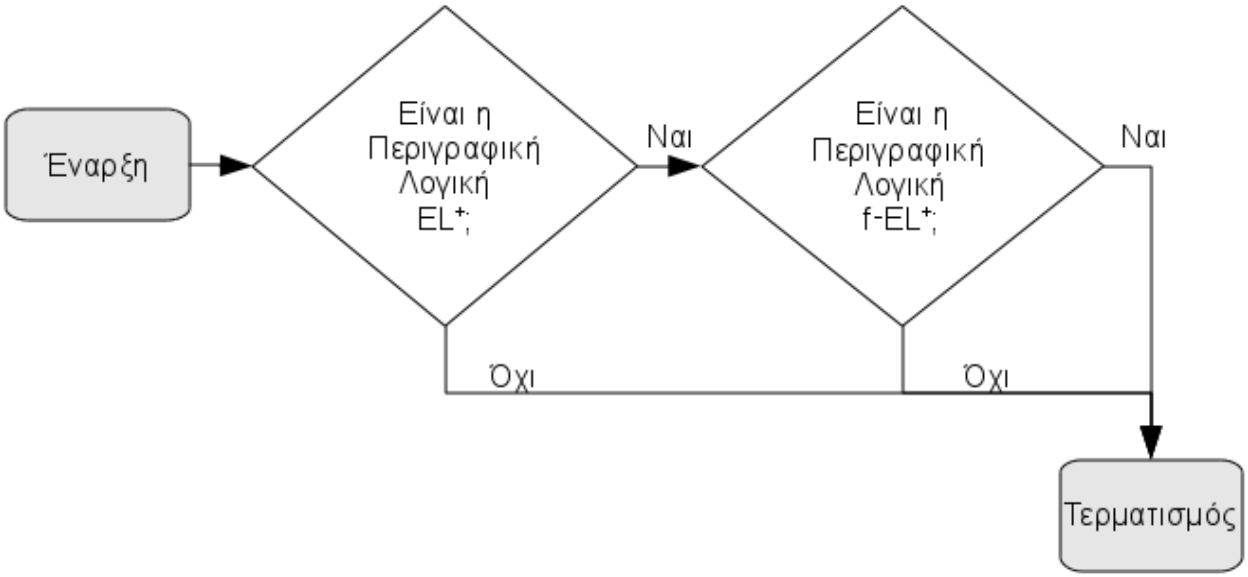
Το υποσύστημα ελέγχου εκφραστικότητας μίας οντολογίας δέχεται ως είσοδο μία οντολογία και επιστρέφει εάν η εκφραστικότητα αυτής μπορεί να γίνει αποδεκτή από τα υπόλοιπα υποσυστήματα. Για τον έλεγχο αυτό πολύ σημαντική είναι η χρήση των μεθόδων και των αντικειμένων που παρέχει το OWL API. Η διαδικασία ελέγχου αναλύεται σε δύο στάδια:

1. **Έλεγχος της γενικής εκφραστικότητας της γλώσσας** με χρήση του ελεγκτή εκφραστικότητας του OWL API (κλάση org.semanticweb.owl.util.DLExpressivityChecker).

2. **Έλεγχος εάν πρόκειται για σαφή ή για ασαφή περιγραφική λογική.** Αυτό διαπιστώνεται αναλόγως της ύπαρξης ή μη επισημειώσεων της μορφής που καθορίζεται στην ενότητα "Απαιτήσεις Δεδομένων Εισόδου".

Αφού διαπιστωθεί η καταλληλότητά ή μη της οντολογίας τότε η λειτουργία του υποσυστήματος σταματά και ο έλεγχος μεταφέρεται στο κεντρικό σύστημα διαχείρισης λειτουργιών.

Το υποσύστημα αυτό είναι ενιαίο, δηλαδή εκτελεί εν γένει τις ίδιες λειτουργίες ανεξαρτήτως του αν η οντολογία είναι καταγεγραμμένη σε Περιγραφική Λογική $\mathcal{EL}^+$ ή $f-\mathcal{EL}^+$. Αυτό δε συμβαίνει στα επόμενα υποσυστήματα, τα οποία όχι μόνο ενδέχεται να εκτελούν διαφορετικές ενέργειες αναλόγως του είδους της Περιγραφικής Λογικής, αλλά μπορεί ακόμα και τα ίδια να αναλύονται σε επιμέρους εξειδικευμένα υποσυστήματα, στοχευμένα είτε στη κλασική $\mathcal{EL}^+$, είτε στην $f-\mathcal{EL}^+$.



Σχήμα 3: *Διάγραμμα Ροής Υποσυστήματος Ελέγχου Εκφραστικότητας*

6.1.2.3 Υποσύστημα Προεπεξεργασίας Οντολογίας

Κύρια λειτουργία του υποσυστήματος προεπεξεργασίας της οντολογίας είναι η αντικατάσταση όλων των αξιωμάτων ισοδυναμίας που εμφανίζονται σε μία οντολογία – είτε αυτά είναι αξιώματα ισοδυναμίας κλάσεων, είτε είναι αξιώματα ισοδυναμίας ρόλων – με το ελάχιστο ισοδύναμο σύνολο αξιωμάτων που αποτελείται μόνο από τα αντίστοιχα αξιώματα υπαγωγής.

Συγκεκριμένα, λαμβάνουν χώρα οι ακόλουθες αντικαταστάσεις:

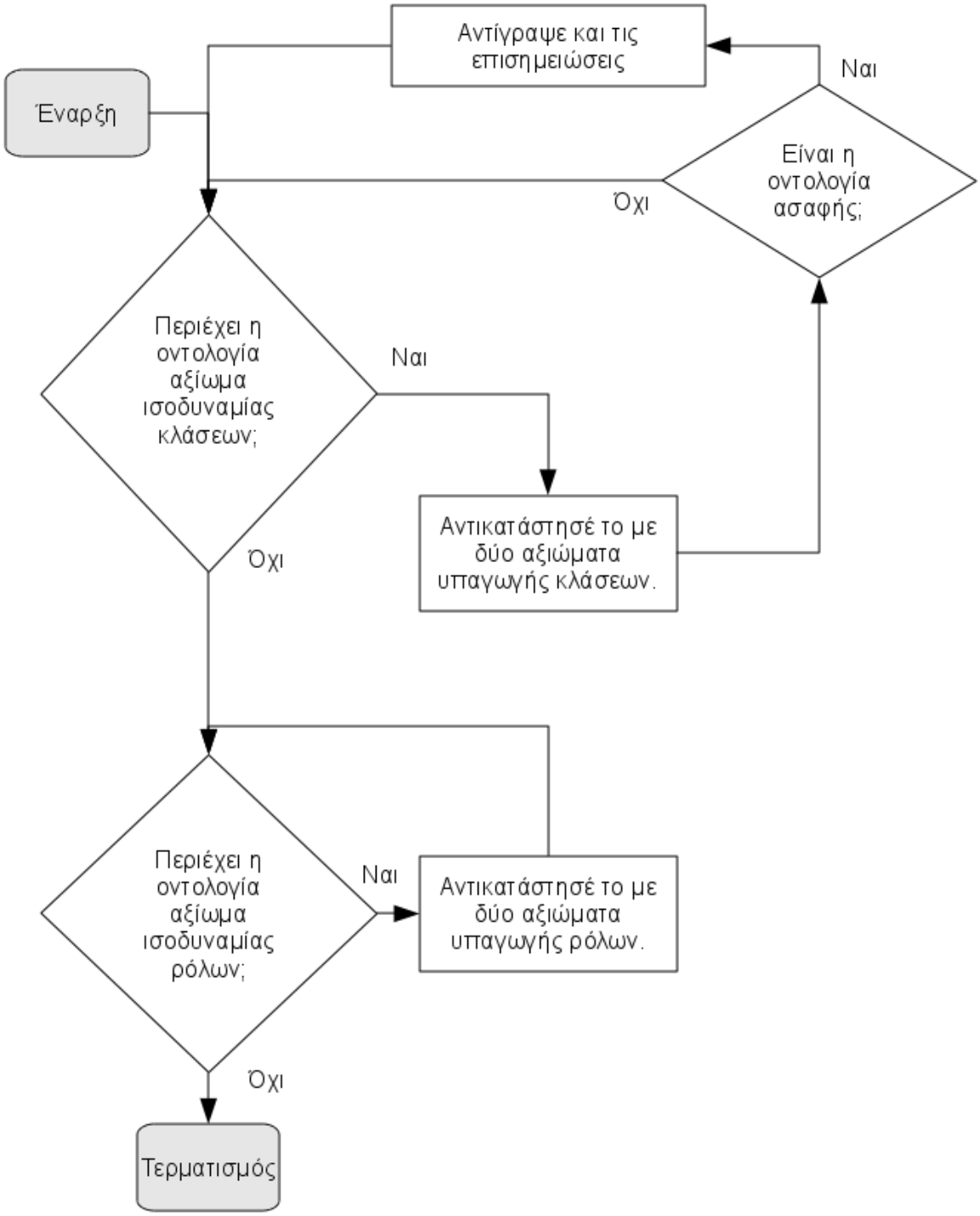
1. $C \equiv D \rightarrow \{C \sqsubseteq D, D \sqsubseteq C\}$, όπου C, D ονοματικές ή μη κλάσεις που συναντώνται στην οντολογία.
2. $r \equiv s \rightarrow \{r \sqsubseteq s, s \sqsubseteq r\}$, όπου r, s ρόλοι που συναντώνται στην οντολογία.

Η σημασιολογία της οντολογίας δεν επηρεάζεται από αυτή τη μετατροπή, η οποία είναι όμως απαραίτητη για την απλοποίηση της υλοποίησης των επομένων σταδίων. Το γεγονός ότι εμφανίζονται λιγότεροι τύποι αξιωμάτων κάνει απλούστερη την υλοποίηση του κώδικα καθώς μειώνει τις διαφορετικές περιπτώσεις που πρέπει αυτός να χειρίζεται. Επίσης, επιτρέπει την κατασκευή μίας υλοποίησης που θα βρίσκεται πλησιέστερα στους παρουσιαζόμενους στη βιβλιο-

γραφία αλγορίθμους και, τέλος, καθιστά ευκολότερη την εποπτεία και την αποσφαλμάτωση του προγράμματος, με συνέπεια τον περιορισμό των σφαλμάτων που υπάρχουν σε αυτό.

Το υποσύστημα αυτό κατά τη λειτουργία του πρέπει να λαμβάνει υπ' όψιν το είδος της οντολογίας που επεξεργάζεται: αν τυχόν αυτή είναι $f - \mathcal{EL}^+$, τότε θα πρέπει να λαμβάνει υπ' όψιν την ύπαρξη αξιωμάτων επισημείωσης και να αντιγράφει τις απαιτούμενες πληροφορίες για τους βαθμούς βεβαιότητας στα νέα αξιώματα που δημιουργεί. Αυτός ακριβώς ο λόγος κάνει το σύστημα να εκτελεί διαφορετικές ενέργειες στην περίπτωση που η οντολογία είναι $f - \mathcal{EL}^+$. Παρ' όλα αυτά, το σύστημα παραμένει ενιαίο και δε χρειάζεται επιμέρους διασπάσεις.

Το διάγραμμα ροής του παρόντος υποσυστήματος απεικονίζεται στο Σχήμα 4.



Σχήμα 4: *Διάγραμμα Ροής Υποσυστήματος Προεπεξεργασίας*

6.1.2.4 Υποσύστημα Κανονικοποίησης Οντολογίας

Η κύρια λειτουργία του υποσυστήματος κανονικοποίησης είναι η εφαρμογή των κανόνων αντικατάστασης αξιωμάτων που παρουσιάζονται στο κεφάλαιο "Σφάλμα: Δεν βρέθηκε η

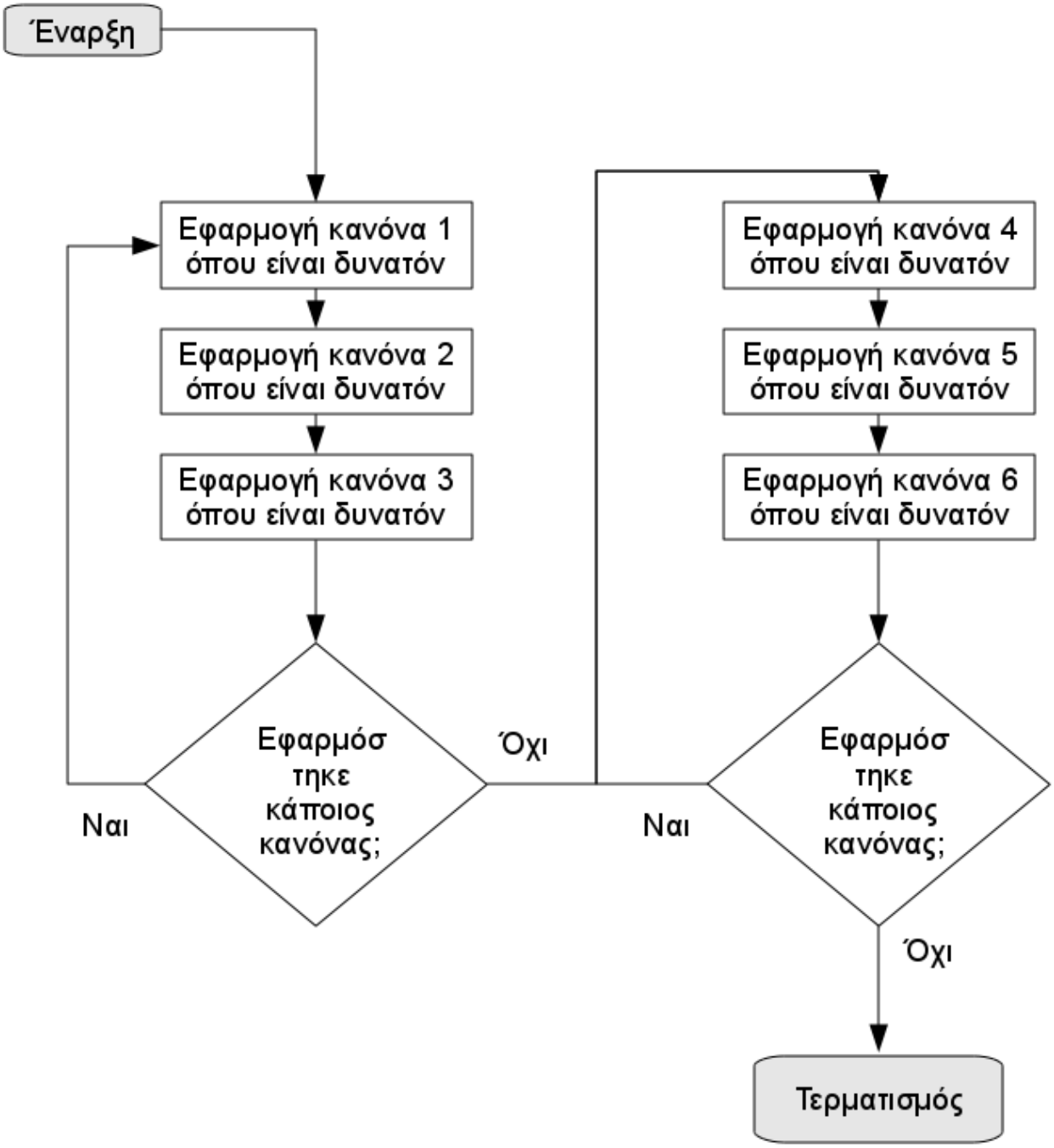
πηγή παραπομπής" για σαφείς ή ασαφείς οντολογίες. Όπως έχει ήδη αναφερθεί, υπάρχουν συνολικά 6 κανόνες οι οποίοι εφαρμόζονται σε δύο στάδια ως εξής:

1. Πρώτον, εξαντλητική εφαρμογή των κανόνων 1 έως 3.
2. Δεύτερον, εξαντλητική εφαρμογή των κανόνων 4 έως 6.

Και αυτό το σύστημα είναι ενιαίο, δε διαφοροποιείται δηλαδή σε επιμέρους υποσυστήματα ανάλογα με το είδος της οντολογίας. Για κάθε περίπτωση έγκυρης Περιγραφικής Λογικής η εφαρμογή ενός κανόνα γίνεται εκτελώντας αρχικά τις απαιτούμενες ενέργειες σα να επρόκειτο για τη κλασική $\mathcal{EL}^+$ και στη συνέχεια, εάν η Περιγραφική Λογική είναι τελικά $f-\mathcal{EL}^+$, εκτελώντας ένα επιπλέον σύνολο ενεργειών που υλοποιεί τη μεταφορά των επισημειώσεων (annotations) – και συνεπώς των βαθμών βεβαιότητας – από το αξίωμα που αντικαθίσταται σε εκείνα που το αντικαθιστούν. Η πρακτική αυτή ακολουθείται για όλους τους κανόνες αντικατάστασης εκτός του πρώτου, ο οποίος δε επηρεάζεται από την ύπαρξη των βαθμολογιών βεβαιότητας στην $f-\mathcal{EL}^+$ και παραμένει ο ίδιος και στις δύο περιπτώσεις. Στο τέλος αυτής τις διαδικασίας έχει εφαρμοσθεί ο κανόνας για την εκάστοτε Περιγραφική Λογική.

Ο τρόπος αυτός σχεδιασμού του αλγορίθμου, δηλαδή η απλή εκτέλεση ορισμένων επιπλέον ενεργειών στην περίπτωση που πρόκειται για ασαφή οντολογία, χρησιμοποιήθηκε με στόχο τη μεγαλύτερη δυνατή επαναχρησιμοποίηση κώδικα και την καλύτερη εποπτεία της λειτουργίας του.

Το διάγραμμα ροής της λειτουργίας του παρόντος υποσυστήματος κανονικοποίησης παρουσιάζεται στο Σχήμα 5, όπου με τον όρο "Εφαρμογή Κανόνα" εννοούμε την εφαρμογή αρχικά των ενεργειών που απαιτούνται για την κλασική $\mathcal{EL}^+$ - οι ενέργειες αυτές είναι κοινές και στις δύο περιπτώσεις – και τη μετέπειτα εφαρμογή της αντιγραφής των επισημειώσεων αν τελικά πρόκειται για οντολογία $f-\mathcal{EL}^+$.



Σχήμα 5: *Διάγραμμα Ροής Υποσυστήματος Κανονικοποίησης*

6.1.2.5 Υποσύστημα Υπολογισμού Συνόλων Αντιστοιχίσεων

Το υποσύστημα υπολογισμού των συνόλων αντιστοιχίσεων (mappings) αποτελεί, μαζί με το υποσύστημα του υπολογισμού του γράφου υπαγωγής που θα εξεταστεί στη συνέχεια, το πολυπλοκότερο υποσύστημα τόσο από πλευράς αρχιτεκτονικής όσο και από πλευράς δυσκολίας υλοποίησης και πλήθους υποστηριζόμενων λειτουργιών.

Για όλα τα προηγούμενα υποσυστήματα έχει αναφερθεί πως η επικάλυψη ανάμεσα στα ομοειδή στάδια των αλγορίθμων συλλογιστικής για σαφείς και ασαφείς Περιγραφικές Λογικές ήταν τόσο σημαντική, που δεν απαιτήθηκε η διάσπαση της υλοποίησης ενός σταδίου σε επιμέρους εξειδικευμένα υποσυστήματα. Όμως, για τον υπολογισμό του συνόλου των αντιστοιχίσεων δεν ισχύει κάτι τέτοιο: η πολυπλοκότητα του υλοποιούμενου αλγορίθμου, σε συνδυασμό με το πλήθος και την ποικιλία των ενεργειών που μπορούν να εκτελεστούν πάνω στο σύνολο των δεδομένων καθιστούν αναγκαία, σε πρώτη φάση, τη διάσπαση του παρόντος υποσυστήματος σε δύο τμήματα:

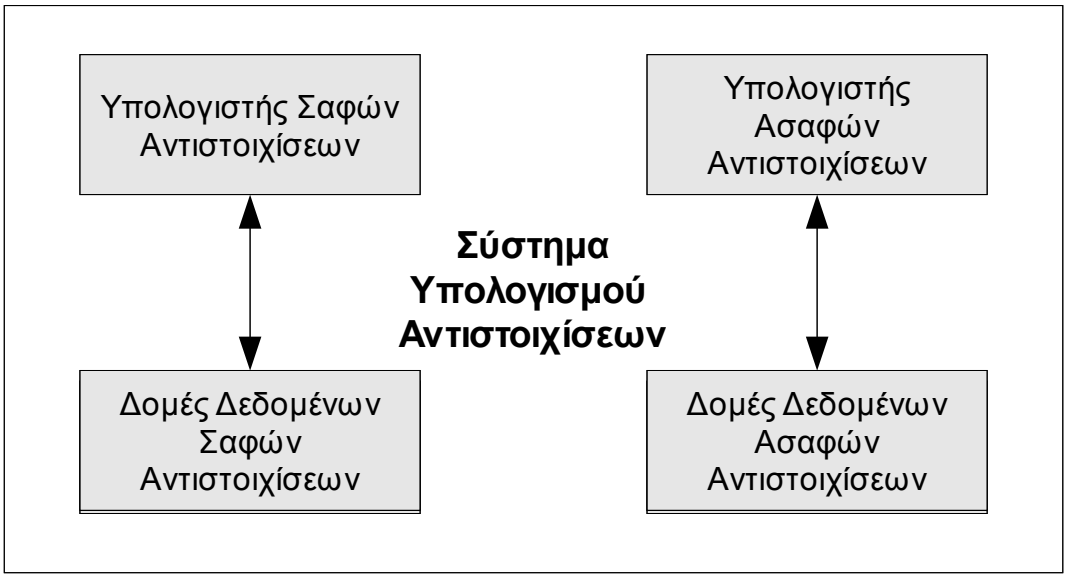
1. Κατ' αρχάς, στο τμήμα εκείνο που θα είναι υπεύθυνο για την εκτέλεση ενεργειών υψηλοτέρου επιπέδου. Εκεί θα υλοποιείται με σχετική αμεσότητα ο αλγόριθμος που παρουσιάζεται στη βιβλιογραφία, χωρίς να λαμβάνεται ιδιαίτερα υπ' όψιν το είδος των χρησιμοποιούμενων δομών δεδομένων ή ο τρόπος αναπαράστασής τους στη μνήμη. Το επίπεδο της υλοποίησης θα είναι κατά το δυνατόν πλησιέστερο στο επίπεδο που χρησιμοποιείται για την περιγραφή του αλγορίθμου συλλογιστικής στη βιβλιογραφία.
2. Κατά δεύτερον, στο τμήμα το οποίο θα υλοποιεί, θα κατέχει, και θα διαχειρίζεται τις δομές δεδομένων που αποθηκεύουν τις αντιστοιχίσεις και όλες τις άλλες βοηθητικές δομές στη μνήμη. Το τμήμα αυτό πρέπει να μπορεί να δέχεται μία εντολή για μία υψηλού επιπέδου λειτουργία από το πρώτο τμήμα και να είναι σε θέση να την υλοποιήσει και να επιστρέψει το κατάλληλο αποτέλεσμα. Ενδεικτικές τέτοιες λειτουργίες είναι όσες παρουσιάζονται στον σχετικό αλγόριθμο, στο κεφάλαιο "Σφάλμα: Δεν βρέθηκε η πηγή παραπομπής".

Η σε πρώτη φάση διάσπαση αυτή σε δύο επιμέρους υποσυστήματα επιτυγχάνει τον στόχο που έχει τεθεί, μειώνοντας την πολυπλοκότητα του συστήματος, βελτιώνοντας την ικανότητα εποπτείας σε αυτό και κυρίως, αποσυνδέοντας τις λειτουργίες που απαιτεί ο αλγόριθμος από τη συγκεκριμένη υλοποίησή τους. Αυτή η τελευταία παράμετρος είναι πολύ σημαντική, καθώς δίνει τη δυνατότητα για αλλαγές και βελτιώσεις στη χαμηλού επιπέδου υλοποίηση του αλγορίθμου χωρίς να επηρεάζεται ο κώδικας που υλοποιεί τον αλγόριθμο σε υψηλό επίπεδο.

Σε δεύτερη φάση, διαπιστώθηκε πως αν και ο αλγόριθμος για τις δύο Περιγραφικές Λογικές $\mathcal{EL}^+$ και $f-\mathcal{EL}^+$ είναι παρόμοιος, δεν είναι αποδοτικό να χρησιμοποιηθεί μία μόνο έκδοσή του και για τις δύο περιπτώσεις. Αφού, όπως ήδη έχει δειχθεί, ο αλγόριθμος συλλογιστικής για την $\mathcal{EL}^+$ είναι μία ειδική περίπτωση αυτού για τη $f-\mathcal{EL}^+$, θα ήταν εφικτό να υλοποιηθεί μόνον αυτός ο αλγόριθμος. Αυτό θα μείωνε ιδιαίτερα το μέγεθος του προγράμματος (πλήθος γραμμών κώδικα). Δυστυχώς όμως, μία τέτοια υλοποίηση δε θα ήταν η βέλτιστη επιλογή καθώς και ο τελικός αλγόριθμος θα παρουσίαζε εν γένει μικρή μείωση της ταχύτητας συλλογιστικής σε σχέση με έναν στοχευμένο στην $\mathcal{EL}^+$, αλλά και η απαιτούμενη μνήμη θα ήταν περισσότερη. Ιδιαίτερα επηρεάζονται πολύπλοκες και μεγάλες οντολογίες, οι οποίες όχι μόνο περιέχουν μεγάλο πλήθος ονοματικών εννοιών και ρόλων, αλλά απαιτούν και πολλές εγγραφές ουράς για την ταξινόμησή τους.

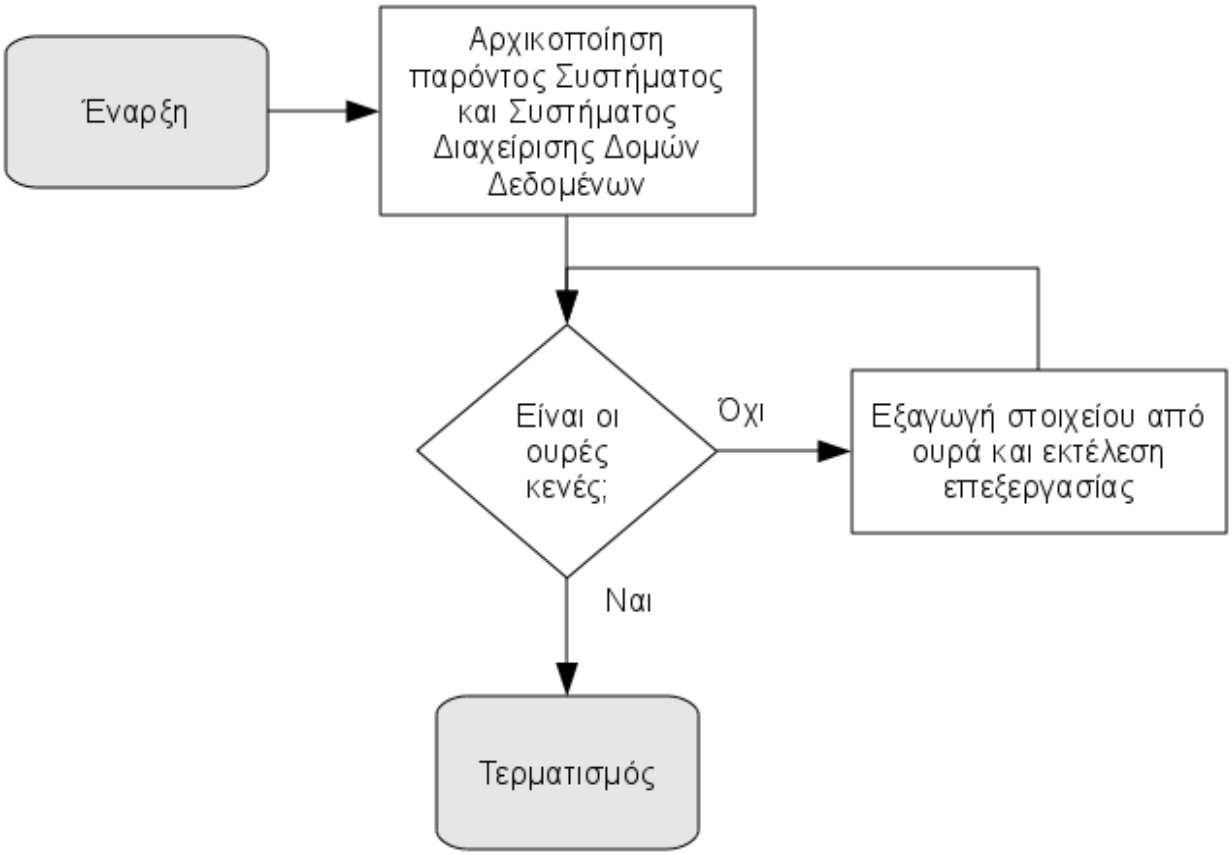
Έτσι, προκειμένου να μη χρησιμοποιείται περισσότερη μνήμη από όση πραγματικά χρειάζεται, αποφασίστηκε και μία περαιτέρω διάσπαση: τελικά χρησιμοποιούνται τέσσερα διαφορετικά υποσυστήματα, δηλαδή ένα ζεύγος για κάθε είδος περιγραφικής λογικής. Κάθε ζεύγος περιέχει ένα υποσύστημα υψηλοτέρου επιπέδου και ένα το οποίο βρίσκεται σε χαμηλότερο επίπεδο και χρησιμοποιείται για το χειρισμό των δομών δεδομένων.

Η αρχιτεκτονική του παρόντος υποσυστήματος παρουσιάζεται στο Σχήμα 6, που εικονίζεται παρακάτω:



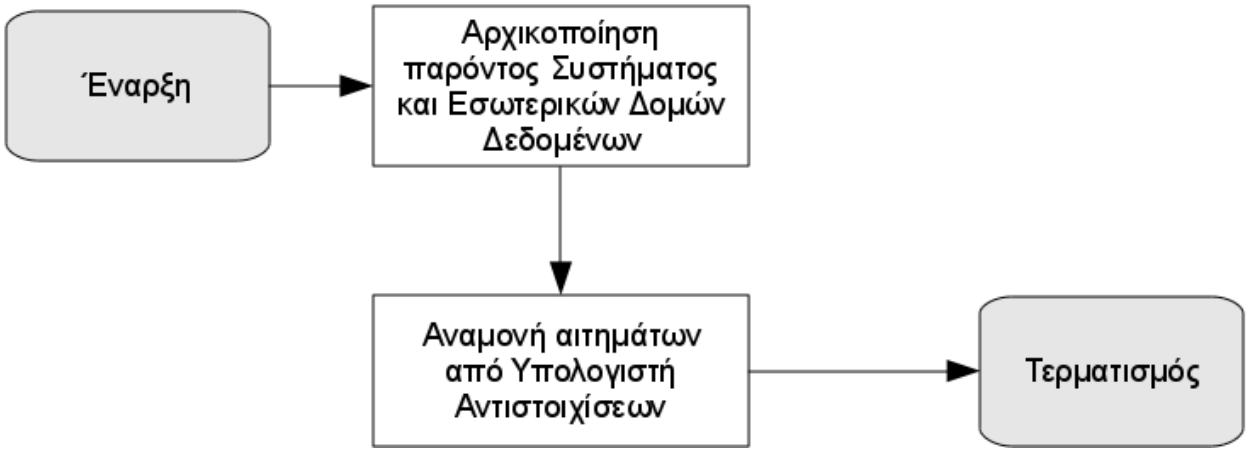
Σχήμα 6: *Αρχιτεκτονική Υποσυστήματος Υπολογισμού Αντιστοιχίσεων*

Το διάγραμμα ροής των ενεργειών που εκτελεί κάθε υποσύστημα Υπολογιστή Αντιστοιχίσεων, είτε σαφών είτε ασαφών, είναι το εξής:



Σχήμα 7: *Διάγραμμα Ροής Ενεργειών Υποσυστήματος Υπολογισμού Αντιστοιχίσεων*

Τέλος, το διάγραμμα ροής των ενεργειών του υποσυστήματος διαχείρισης δομών δεδομένων είναι το ακόλουθο:



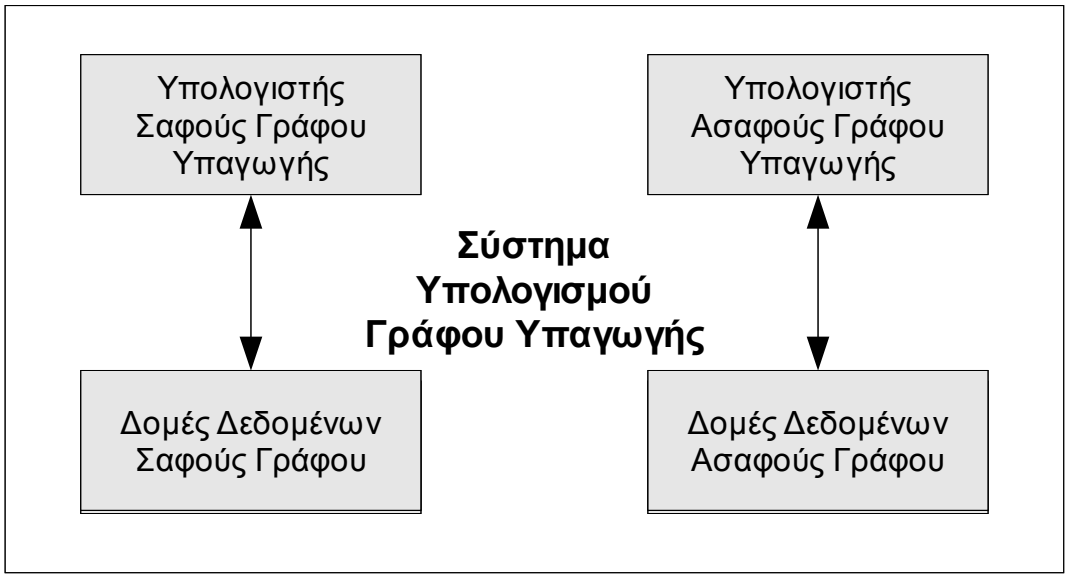
Σχήμα 8: *Διάγραμμα Ροής Ενεργειών Υποσυστήματος Διαχείρισης Δομών Δεδομένων Υπολογιστή Αντιστοιχίσεων*

6.1.2.6 Υποσύστημα Υπολογισμού Γράφου Υπαγωγής

Το υποσύστημα αυτό χρησιμοποιείται μόνο σε περίπτωση που οι άμεσες σχέσεις υπαγωγής που υπολογίζει είναι απαραίτητες για την απάντηση των ερωτημάτων του χρήστη.

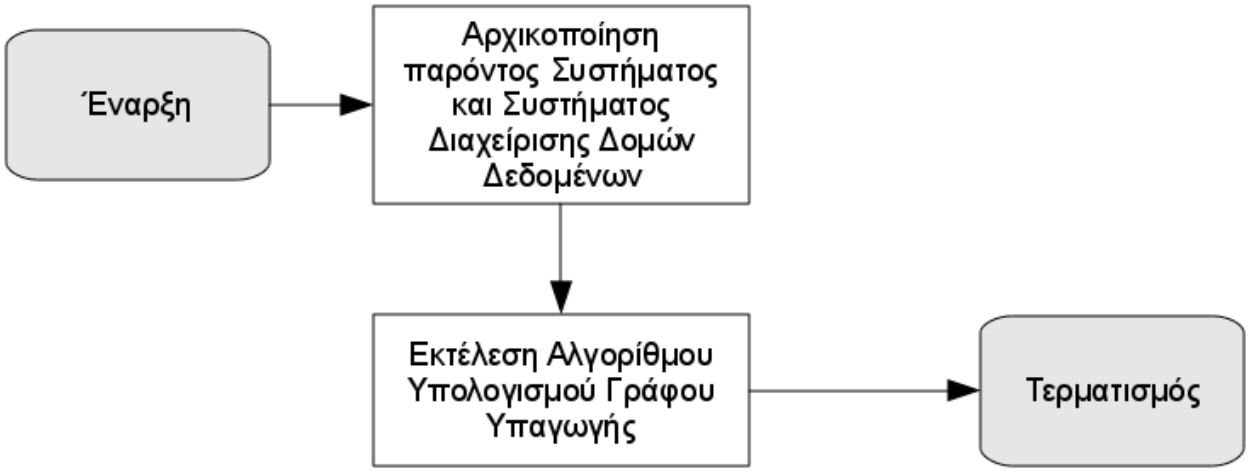
Όπως ακριβώς και το υποσύστημα υπολογισμού αντιστοιχίσεων, έτσι και το υποσύστημα υπολογισμού του γράφου υπαγωγής έχει διασπασθεί και αυτό σε τέσσερα υποσυστήματα. Ο τρόπος διάσπασης είναι ο ίδιος, υφίστανται δηλαδή και εδώ δύο ζεύγη υποσυστημάτων, καθένα εκ των οποίων εξειδικεύεται είτε στην κλασική $\mathcal{EL}^+$ είτε στη $f-\mathcal{EL}^+$. Κάθε ζεύγος αποτελείται από ένα υποσύστημα το οποίο υλοποιεί τον αλγόριθμο με λειτουργίες υψηλού επιπέδου και από ένα άλλο υποσύστημα, το οποίο είναι υπεύθυνο για την αρχικοποίηση και διαχείριση των δομών δεδομένων που αποθηκεύουν τις απαραίτητες πληροφορίες στη μνήμη. Το δεύτερο αυτό χαμηλοτέρου επιπέδου υποσύστημα ουσιαστικά υλοποιεί τα υψηλού επιπέδου αιτήματα του πρώτου υποσυστήματος.

Η αρχιτεκτονική του παρόντος υποσυστήματος παρουσιάζεται στο Σχήμα 9:



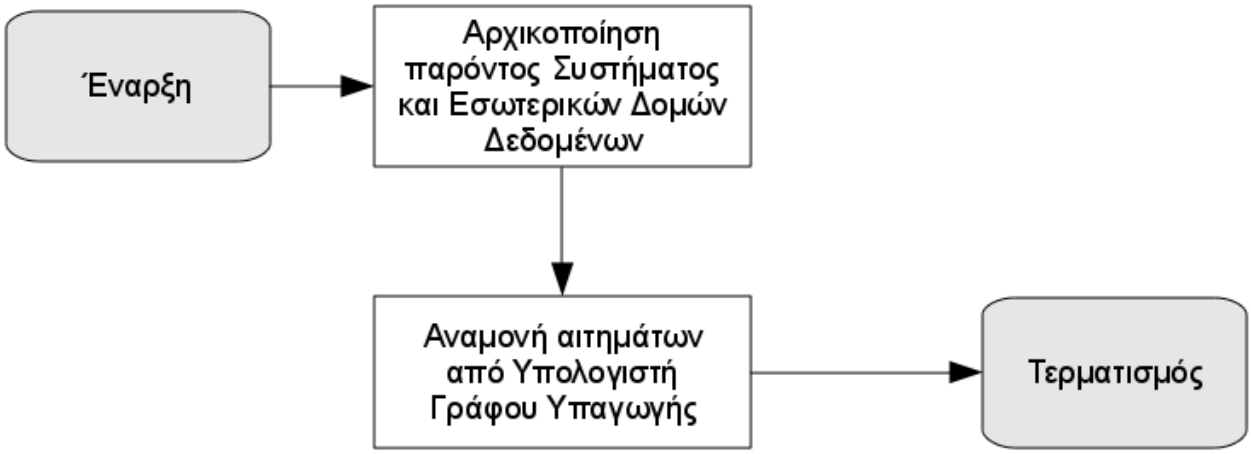
Σχήμα 9: *Αρχιτεκτονική Υποσυστήματος Υπολογισμού Γράφου Υπαγωγής*

Κάθε τμήμα υπολογιστή γράφου υπαγωγής ουσιαστικά αρχικοποιεί το υποσύστημα διαχείρισης δομών δεδομένων και στη συνέχεια εκτελεί τον αλγόριθμο που παρουσιάζεται στη βιβλιογραφία:



Σχήμα 10: *Διάγραμμα Ροής ενεργειών υποσυστήματος Υπολογιστή Γράφου Υπαγωγής*

Ακόμα, οι ενέργειες κάθε τμήματος διαχείρισης των δομών δεδομένων στις οποίες αποθηκεύονται οι απαραίτητες πληροφορίες για τον υπολογισμό του γράφου υπαγωγής περιγράφονται από το ακόλουθο διάγραμμα ροής:



Σχήμα 11: *Διάγραμμα ροής ενεργειών υποσυστήματος διαχείρισης δομών δεδομένων γράφου υπαγωγής*

6.2 Διεπαφή με χρήστη

6.2.1 Διεπαφή ως ανεξάρτητο πρόγραμμα

Όπως έχει ήδη αναφερθεί και στην ενότητα "Σφάλμα: Δεν βρέθηκε η πηγή παραπομπής", η βασική υπηρεσία συλλογιστικής που προσφέρεται για τις $\mathcal{EL}^+$ και $f - \mathcal{EL}^+$ είναι η υπηρεσία ταξινόμησης (classification) μία οντολογίας. Η υπηρεσία αυτή είναι και η μόνη η οποία παρέχεται από την υλοποιηθείσα μηχανή συλλογιστικής. Όμως, παρά το γεγονός ότι η υπηρεσία είναι μόνο μία, οι διαφορετικές δυνατότητες εκτέλεσης ερωτημάτων και εμφάνισης αποτελεσμάτων είναι πολύ περισσότερες. Βεβαίως, πρέπει να επισημανθεί πως η μηχανή συλλογιστικής **δεν υποστηρίζει μόνο την απάντηση ερωτημάτων υπαγωγής ονοματικών κλάσεων**. Έχουν υλοποιηθεί οι κατάλληλες επεκτάσεις έτσι ώστε να υποστηρίζεται ο **έλεγχος υπαγωγής ή ισοδυναμίας συνθέτων εννοιών** τόσο σε σαφείς όσο και σε ασαφείς οντολογίες.

Έτσι, παρέχονται οι ακόλουθες δυνατότητες στο χρήστη όσον αφορά την εξαγωγή των αποτελεσμάτων της εξαγωγής:

1. **Απλή ταξινόμηση της οντολογίας** με άμεση έξοδο μετά το πέρας αυτής. Αυτή η δυνατότητα έχει νόημα κυρίως για έλεγχο της ορθής λειτουργίας της μηχανής συλλογιστικής και της σύνταξης του αρχείου καταγραφής της οντολογίας εισόδου, ενώ επίσης χρησιμοποιείται και για τη μέτρηση του χρόνου που απαιτείται για ταξινόμηση διαφόρων οντολογιών.
2. **Ταξινόμηση της οντολογίας και στο τέλος εμφάνιση στο χρήστη της δυνατότητας να εισάγει συμβολοσειρές**. Κάθε εισαχθείσα συμβολοσειρά συγκρίνεται με τα ονόματα των κλάσεων της οντολογίας και αν περιέχεται σε κάποιο από αυτά, τότε εκτυπώνονται οι πληροφορίες που προέκυψαν με το πέρας του αλγορίθμου υπολογισμού του συνεκτικού ακυκλικού γράφου υπαγωγής. Συγκεκριμένα, εμφανίζονται τα ονόματα των κλάσεων-γονέων που υπάγουν άμεσα τη δοθείσα κλάση, των ισοδυνάμων κλάσεων και των κλάσεων-παιδιών, οι οποίες υπάγονται άμεσα από αυτήν. Στην περίπτωση που πρόκειται για

οντολογία $f - \mathcal{EL}^+$ εμφανίζεται για κάθε καταχώριση και ο αντίστοιχος βαθμός βεβαιότητας.

3. **Δυνατότητα ορισμού ζευγών ονομάτων κλάσεων, οι οποίες θα ελεγχθούν για υπαγωγή ή για ισοδυναμία**. Αποτέλεσμα της διαδικασίας αυτής είναι η εκτύπωση, για κάθε ζεύγος, του βαθμού ισχύος του ερωτήματος. Όπως είναι φανερό, στην περίπτωση της κλασικής $\mathcal{EL}^+$ το αποτέλεσμα θα ανήκει στο σύνολο $\{0, 1\}$, ενώ στην περίπτωση της $f - \mathcal{EL}^+$ θα βρίσκεται στο διάστημα $[0, 1]$.

4. **Δυνατότητα καθορισμού ενός αρχείου, τοπικού ή απομακρυσμένου, το οποίο θα περιέχει τα επιθυμητά ερωτήματα του χρήστη** σε γλώσσα OWL 2 και με σύνταξη κάποια από τις αναγνωριζόμενη από το OWL API, όπως για παράδειγμα η OWL/XML. Τα ερωτήματα αυτά θεωρούνται έγκυρα εφ' όσον είναι ερωτήματα υπαγωγής ή ισοδυναμίας κλάσεων. Οι κλάσεις δεν είναι απαραίτητο να είναι ονοματικές: υποστηρίζεται η δυνατότητα συλλογιστικής και ελέγχου υπαγωγής ή ισοδυναμίας μεταξύ πολύπλοκων εννοιών.

   Στην περίπτωση αυτή οι απαντήσεις στα ερωτήματα μπορούν να παρουσιαστούν με δύο τρόπους:

   i. **Με απλή εκτύπωση κάθε αξιώματος και του βαθμού με τον οποίο αυτό ισχύει.** Ισχύουν οι ίδιες συμβάσεις για τον ασαφή βαθμό όπως και στην περίπτωση της δυνατότητας 3.

   ii. **Με προσθήκη αξιωμάτων επισήμανσης σε κάθε αξίωμα της αρχικής οντολογίας.** Συγκεκριμένα, προστίθεται ο βαθμός βεβαιότητας ακολουθώντας το πρότυπο που παρουσιάστηκε στην ενότητα "Απαιτήσεις Δεδομένων Εισόδου". Η οντολογία αυτή αποθηκεύεται στο τοπικό σύστημα αρχείων και μπορεί στη συνέχεια να προσπελαστεί από τον χρήστη ή κάποιο πρόγραμμα – πελάτη. Για τον περιορισμό του μεγέθους της τελικής οντολογίας, ο βαθμός προστίθεται ως επισημείωση μόνο εάν είναι μικρότερος της μονάδας. Η τακτική αυτή είναι συμβατή με την παραδοχή που ακολουθείται να θεωρείται η ανυπαρξία βαθμού επισημείωσης ως ένδειξη ισχύος του σχετικού αξιώματος με απόλυτη βεβαιότητα.

5. **Δυνατότητα εξαγωγής των ονομάτων των κλάσεων-γονέων, των ισοδυνάμων κλάσεων και των κλάσεων-παιδιών για κάθε ονοματική κλάση της οντολογίας σε αρχείο.** Οι πληροφορίες που εξάγονται είναι οι ίδιες που παρουσιάζονται στον χρήστη στην περίπτωση της δυνατότητας 2. Σε περίπτωση που πρόκειται για ασαφή οντολογία εμφανίζονται και πληροφορίες βαθμών βεβαιότητας.

6. **Δυνατότητα προσθήκης των αξιωμάτων ορισμού της υπολογισθείσας ιεραρχίας των κλάσεων στην αρχική οντολογία.** Είναι δυνατόν λοιπόν να προστεθούν στην αρχική οντολογία όλα τα υπολογιζόμενα αξιώματα υπαγωγής και ισοδυναμίας κλάσεων, έτσι ώστε η ιεραρχία τους να είναι άμεσα ορατή σε κάποιο γραφικό εργαλείο ανάλυσης και επεξεργασίας πληροφοριών, όπως το Protégé.

Επίσης, προσφέρεται η **δυνατότητα εξαγωγής της κανονικοποιημένης οντολογίας**. Η δυνατότητα αυτή είναι χρήσιμη κυρίως για τον έλεγχο της λειτουργίας της μηχανής συλλογιστικής και δεν προσφέρει κάτι στον τελικό χρήστη.

6.2.2 Διεπαφή ως υπηρεσία

Λόγω των περιορισμών που υφίστανται σε μία απομακρυσμένη σύνδεση, αλλά και το γεγονός ότι η δυνατότητα διεπαφής ως υπηρεσία υλοποιήθηκε κατά κύριο λόγο για την εξυπηρέτηση αιτημάτων προγραμμάτων – πελατών και όχι για την εξυπηρέτηση ενός ανθρώπου – χρήστη άμεσα, προσφέρεται σε αυτή μόνο ένα υποσύνολο των λειτουργιών της πλήρους μηχανής συλλογιστικής. Έτσι, εάν κάποια λειτουργία η οποία δεν προσφέρεται είναι σημαντική για κάποιον χρήστη ή ο χρήστης αυτός ενδιαφέρεται σημαντικά για τον ρυθμό διεκπεραίωσης (throughput) των αιτημάτων του τότε προτείνεται η εγκατάσταση της μηχανής συλλογιστικής σε κάποιο τοπικό μηχάνημα και η χρήση της από την τοπική διεπαφή.

Έτσι, οι υπηρεσίες οι οποίες παρέχονται διαμέσου της απομακρυσμένης διεπαφής είναι οι ακόλουθες:

1. Ανάκτηση οντολογίας είτε από το τοπικό σύστημα αρχείων είτε μέσω του πρωτοκόλλου HTTP και ταξινόμησή της.
2. Απάντηση ερωτημάτων ισοδυναμίας ή υπαγωγής ονοματικών κλάσεων.

Η διεκπεραίωση των αιτημάτων για την υπηρεσία απάντησης ερωτημάτων ισοδυναμίας ή υπαγωγής ονοματικών κλάσεων δεν απαιτεί τον υπολογισμό του συνεκτικού ακυκλικού γράφου υπαγωγής (subsumption DAG), επομένως ο υπολογισμός του δε λαμβάνει χώρα σε οποιαδήποτε προσπάθεια ταξινόμησης οντολογίας μέσω της απομακρυσμένης υπηρεσίας.

7. Περιγραφή Υλοποίησης Συστήματος

Στο προηγούμενο κεφάλαιο, "Περιγραφή Σχεδίασης Συστήματος", παρουσιάστηκαν οι απαιτήσεις και οι προδιαγραφές του προγράμματος, οι λειτουργίες που αυτό υποστηρίζει, καθώς και η γενική αρχιτεκτονική του. Στο κεφάλαιο αυτό θα παρουσιαστεί η συγκεκριμένη υλοποίηση του συστήματος, κάνοντας αναφορά σε κλάσεις και μεθόδους του κώδικα.

Για την υλοποίηση του συστήματος χρησιμοποιήθηκαν 25 κλάσεις Java, οι οποίες αντιστοιχούν σε λίγο περισσότερο από 6.000 γραμμές κώδικα. Οι κλάσεις αυτές έχουν εν γένει χωρισθεί σε 3 διαφορετικά πακέτα Java (Java packages):

1. Πρώτον, το πακέτο FELPlusReasoner (Fuzzy $\mathcal{EL}^+$ Reasoner), το οποίο περιέχει τη Main συνάρτηση του προγράμματος, τις δύο υποστηριζόμενες διεπαφές και μεθόδους που υλοποιούν σε υψηλό επίπεδο τον αλγόριθμο συλλογιστικής ή άλλες βοηθητικές λειτουργίες.
2. Δεύτερον, το πακέτο Datatypes, στο οποίο ορίζονται σύνθετες δομές δεδομένων που έχουν χρησιμοποιηθεί για την υλοποίηση του συστήματος. Εδώ επίσης υλοποιούνται και οι λειτουργίες που υποστηρίζονται σε αυτές.
3. Το πακέτο Utilities που περιέχει κλάσεις οι οποίες υλοποιούν λειτουργίες χρήσιμες για την εκτέλεση του προγράμματος, οι οποίες όμως μπορούν να θεωρηθούν γενικές μιας και δεν έχουν άμεση σχέση με χειρισμό οντολογιών ή συλλογιστική.

Από τα παραπάνω γίνεται αντιληπτό ότι το κύριο τμήμα του αλγορίθμου συλλογιστικής βρίσκεται στο πακέτο FELPlusReasoner, ενώ οι χαμηλοτέρου επιπέδου υλοποιήσεις μεθόδων που το κύριο τμήμα καλεί βρίσκονται στο Datatypes.

Σημαντικό είναι να αναφερθεί πως πληροφορίες για το ρόλο όλων των κλάσεων, μεταβλητών και μεθόδων έχουν καταγραφεί και σε μορφή Javadoc. Οι πληροφορίες αυτές μπορούν να βρεθούν στα αρχεία του κώδικα του προγράμματος.

7.1 Πακέτο FELPlusReasoner

Το πακέτο FELPlusReasoner περιέχει τις ακόλουθες κλάσεις, οι οποίες παρουσιάζονται με αλφαβητική σειρά:

1. **CrispDAGComputer:** Η κλάση αυτή αναλαμβάνει τον υπολογισμό του γράφου υπαγωγής (subsumption DAG) για μία κλασική $\mathcal{EL}^+$ οντολογία, αφ' ότου έχει γίνει ο υπολογισμός των αντιστοιχίσεων S και R από την κλάση CrispMappingsComputer.
2. **CrispMappingsComputer:** Η κλάση αυτή αναλαμβάνει τον υπολογισμό των αντιστοιχίσεων S και R για μία κανονικοποιημένη $\mathcal{EL}^+$ οντολογία.
3. **FuzzyDAGComputer:** Η κλάση αυτή αναλαμβάνει τον υπολογισμό του γράφου υπαγωγής (subsumption DAG) για μία $f-\mathcal{EL}^+$ οντολογία, αφ' ότου έχει γίνει ο υπολογισμός των αντιστοιχίσεων S και R από την κλάση FuzzyMappingsComputer.
4. **FuzzyMappingsComputer:** Η κλάση αυτή αναλαμβάνει τον υπολογισμό των αντιστοιχίσεων S και R για μία κανονικοποιημένη $f-\mathcal{EL}^+$ οντολογία.
5. **Main:** Η κύρια κλάση, απ' όπου ξεκινά η εκτέλεση του προγράμματος.

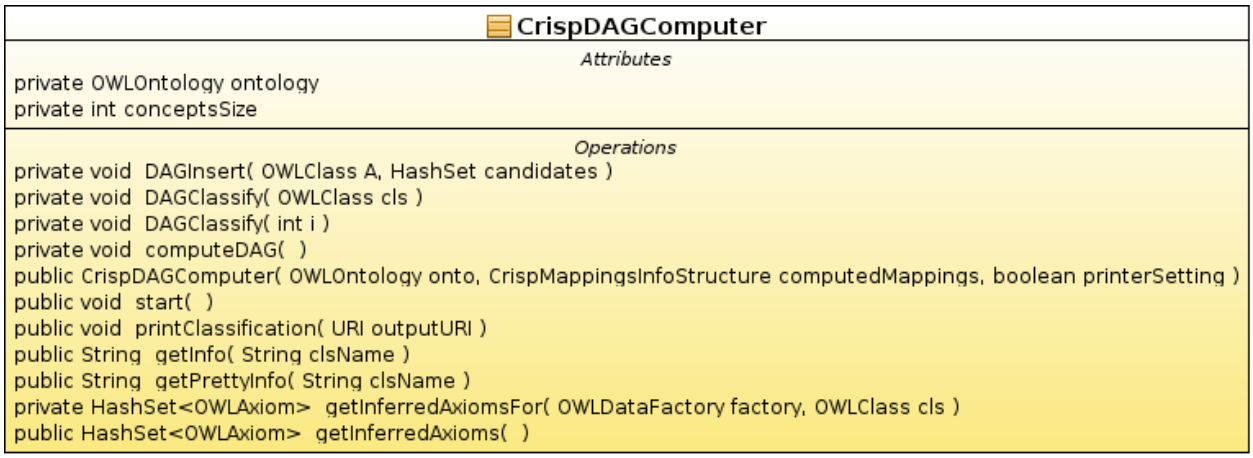
6. **Normaliser:** Η κλάση αυτή κανονικοποιεί μία $\mathcal{EL}^+$ ή $f-\mathcal{EL}^+$ οντολογία με εξαντλητική εφαρμογή 6 κανόνων σε δύο στάδια (3 κανόνες σε κάθε στάδιο), όπως ακριβώς και ο θεωρητικός αλγόριθμος.
7. **OntologyWorker:** Η κλάση OntologyWorker δίνει τη δυνατότητα εκτέλεσης διαφόρων λειτουργιών πολύ υψηλού επιπέδου σε μία οντολογία, όπως η ταξινόμηση (classification).
8. **RoleHierarchyReflexiveTrasitiveClosureComputer:** Η κλάση αυτή χρησιμοποιείται για τον υπολογισμό του ανακλαστικού - μεταβατικού κλεισίματος του συνόλου των αξιωμάτων ιεραρχίας ρόλων της οντολογίας.
9. **ThreadedRequestHander:** Η κλάση αυτή αναλαμβάνει την εξυπηρέτηση μίας εισερχόμενης απομακρυσμένης σύνδεσης κάποιου πελάτη στη μηχανή συλλογιστικής.

Στη συνέχεια θα περιγραφούν οι κλάσεις αυτές και οι βασικές τους μέθοδοι με περισσότερη λεπτομέρεια. Όμως, η περιγραφή αυτή δε θα είναι πλήρης: λεπτομερέστερη περιγραφή γίνεται στο παρεχόμενο Javadoc του προγράμματος. Περιγραφή τέτοιας λεπτομέρειας στο παρόν κεφάλαιο ξεφεύγει από το επίπεδο της αναφοράς διπλωματικής εργασίας.

7.1.1 Κλάση CrispDAGComputer

Η κλάση αυτή χρησιμοποιείται για τον υπολογισμό του συνεκτικού ακυκλικού γράφου υπαγωγής μίας κλασικής $\mathcal{EL}^+$ οντολογίας. Υλοποιεί τον θεωρητικό αλγόριθμο που παρουσιάζεται στην ενότητα "Υπολογισμός Άμεσων Σχέσεων Υπαγωγής" (σελ. 30). Αντιστοιχεί στον "Υπολογιστή Σαφούς Γράφου Υπαγωγής", όπως αυτός παρουσιάζεται στο Σχήμα 9 (σελ. 59) της ενότητας "Υποσύστημα Υπολογισμού Γράφου Υπαγωγής".

Το UML διάγραμμα της κλάσης είναι το ακόλουθο:



Διάγραμμα 1: *Διάγραμμα UML της κλάσης FELPlusReasoner.CrispDAGComputer*

Υπάρχουν οι ακόλουθες αντιστοιχίσεις ανάμεσα στις διαδικασίες του αναφέρονται στον θεωρητικό αλγόριθμο και στις μεθόδους που υλοποιεί η παρούσα κλάση:

1. Η θεωρητική διαδικασία $\textit{υπολογισμός}-\Sigma A\Gamma()$ υλοποιείται από τη μέθοδο computeDAG.

2. Η $\Sigma A\Gamma - \tau\alpha\xi\iota\nu\acute{o}\mu\eta\sigma\eta(A)$ του θεωρητικού αλγορίθμου υλοποιείται από τη μέθοδο DAGClassify.
3. Τέλος, η $\Sigma A\Gamma - \epsilon\iota\sigma\alpha\gamma\omega\gamma\acute{\eta}(A, \upsilon\pi o\psi\acute{\eta}\varphi\iota o\iota)$ υλοποιείται από τη μέθοδο DAGInsert της κλάσης .

Η κλάση αυτή δέχεται το CrispMappingsInfoStructure input, στο οποίο αποθηκεύονται τα υπολογισθέντα σύνολα αντιστοιχίσεων S και R από το προηγούμενο στάδιο της διαδικασίας, ενώ χρησιμοποιεί το CrispDAGInfoStructure workingDataSet, το οποίο και αποτελεί το αντικείμενο στο οποίο αποθηκεύονται τα δεδομένα του γράφου υπαγωγής επί των οποίων εργάζεται. Επίσης, ένα CrispDAGInfoStructure είναι σε θέση να εκτελέσει ορισμένες λειτουργίες πάνω στα δεδομένα που αποθηκεύει.

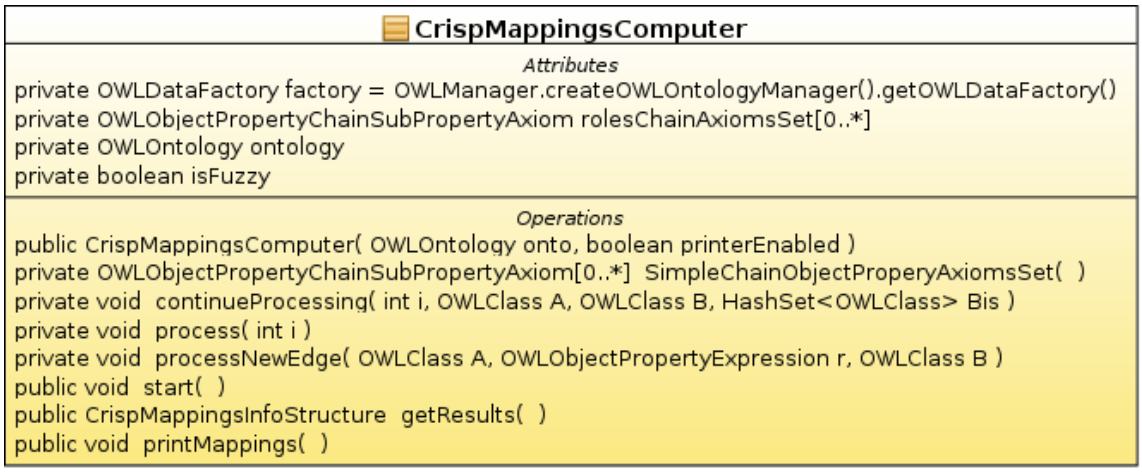
Οι υπόλοιπες μέθοδοι υλοποιούν βοηθητικές λειτουργίες, εκ των οποίων οι σημαντικότερες είναι οι εξής:

1. Η printClassification(java.net.URI outputURI) εκτυπώνει τις υπολογισθείσες πληροφορίες ταξινόμησης (κλάσεις-γονείς, ισοδύναμες κλάσεις και κλάσεις-παιδιά) για όλες τις ονοματικές κλάσεις μίας οντολογίας σε κάποιο αρχείο.
2. Οι getPrettyInfo(java.lang.String clsName) και getInfo(java.lang.String clsName) επιστρέφουν σε μορφή κατάλληλη για εκτύπωση πληροφορίες ταξινόμησης για μία συγκεκριμένη κλάση.
3. Η getInferredAxioms() επιστρέφει το σύνολο των αξιωμάτων που πρέπει να εισαχθούν στην οντολογία, έτσι ώστε αυτή να περιέχει καταγεγραμμένες όλες τις υπολογιζόμενες πληροφορίες ταξινόμησης (υπαγωγές, ισοδυναμίες) για κάθε κλάση.

7.1.2 Κλάση CrispMappingsComputer

Η κλάση CrispMappingsComputer χρησιμοποιείται για τον υπολογισμό των συνόλων αντιστοιχίσεων S για κάθε ονοματική έννοια και R για κάθε ονοματικό ρόλο μίας κλασικής $\mathcal{EL}^+$ οντολογίας. Ο αλγόριθμος που υλοποιείται αντιστοιχεί στον θεωρητικό αλγόριθμο που παρουσιάζεται στην ενότητα “Βελτιωμένος Αλγόριθμος Υπολογισμού Αντιστοιχίσεων” (σελ. 28). Αντιστοιχεί στον “Υπολογιστή Σαφών Αντιστοιχίσεων” που παρουσιάζεται στο Σχήμα 6 (σελ. 57) της ενότητας “Υποσύστημα Υπολογισμού Συνόλων Αντιστοιχίσεων”.

Το UML διάγραμμα της κλάσης αυτής παρουσιάζεται παρακάτω:



Διάγραμμα 2: *Διάγραμμα UML της κλάσης FELPlusReasoner.CrispMappingsComputer*

Ανάμεσα στις διαδικασίες που αναφέρονται στον θεωρητικό αλγόριθμο και στις μεθόδους που υλοποιεί η παρούσα κλάση υφίστανται οι ακόλουθες αντιστοιχίσεις:

1. Η διαδικασία $\textit{επεξεργασία}(A, X)$ υλοποιείται από τις μεθόδους process(int i) και continueProcessing(int i, org.semanticweb.owl.model.OWLClass A,

   org.semanticweb.owl.model.OWLClass B,

   java.util.HashSet<org.semanticweb.owl.model.OWLClass> Bis).

   Η διαδικασία $\textit{επεξεργασία}(A, X)$ διαχωρίστηκε σε δύο μεθόδους προκειμένου να είναι ευκολότερη η επαναχρησιμοποίηση και συνεπώς η εποπτεία του κώδικα. Έτσι, η μέθοδος process καλεί την continueProcessing με αποτέλεσμα οι δύο μαζί να διεκπεραιώνουν τις ενέργειες που απαιτεί ο θεωρητικός αλγόριθμος.

2. Η διαδικασία $\textit{επεξεργασία}-\textit{νέας}-\textit{ακμής}(A, r, B)$ υλοποιείται από τη μέθοδο processNewEdge(org.semanticweb.owl.model.OWLClass A,

   org.semanticweb.owl.model.OWLObjectPropertyExpression r,

   org.semanticweb.owl.model.OWLClass B).

Η κλάση αυτή δέχεται την κανονικοποιημένη οντολογία από το προηγούμενο στάδιο της κανονικποίησης, η οποία και για να μπορεί να προσπελασθεί τοπικά αποθηκεύεται στη μεταβλητή org.semanticweb.owl.model.OWLOntology ontology. Επίσης, χρησιμοποιεί το αντικείμενο CrispMappingsInfoStructure workingDataSet προκειμένου να αποθηκεύει τα δεδομένα στα οποία εργάζεται. Ένα CrispMappingsInfoStructure είναι σε θέση να εκτελέσει ορισμένες λειτουργίες πάνω στα δεδομένα που αποθηκεύει.

Σημαντικές άλλες λειτουργίες οι οποίες υλοποιούνται από μεθόδους της κλάσης αυτής είναι οι ακόλουθες:

1. Πρώτον, η μέθοδος SimpleChainObjectProperyAxiomsSet(), η οποία συγκεντρώνει σε ένα σύνολο όλα τα αξιώματα τα οποία είναι αλυσίδες ρόλων με δύο στοιχεία στην υποκλάση, δηλαδή όλα τα αξιώματα μορφής $r \circ s \sqsubseteq u$, όπου r, s και u ονοματικοί ρόλοι. Στη συνέχεια επιστρέφει το σύνολο αυτό. Η μέθοδος αυτή χρησιμοποιείται από την processNewEdge.

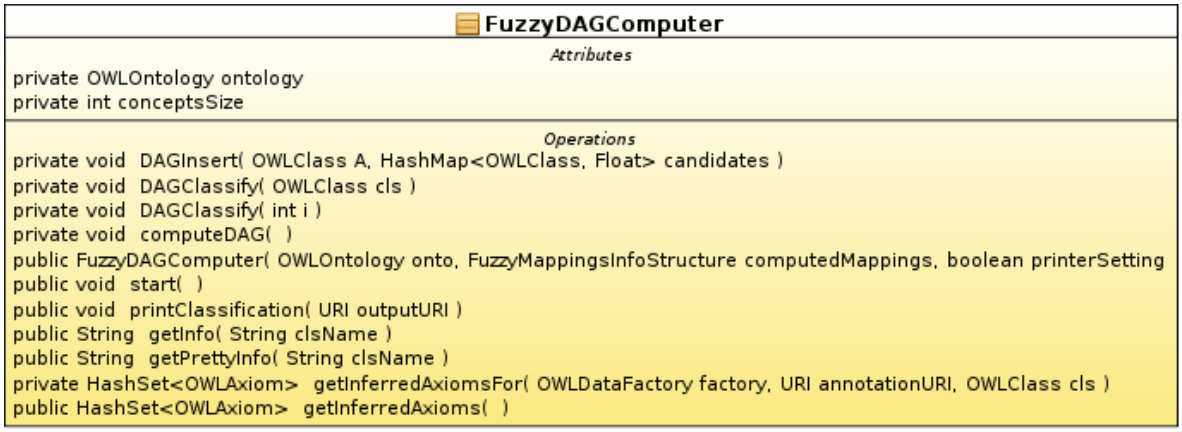
2. Δεύτερον, η μέθοδος getResults(), η οποία επιτρέπει να λάβουμε το CrispMappingsInfoStructure στο οποίο αποθηκεύονται τα αποτελέσματα. Η λειτουργία αυτή χρησιμοποιείται από το κεντρικό σύστημα διαχείρισης των λειτουργιών της μηχανής, το οποίο και υλοποιείται στην κλάση OntologyWorker, προκειμένου αυτό να λάβει τα υπολογισθέντα σύνολα αντιστοιχίσεων. Στη συνέχεια τα σύνολα αυτά θα δοθούν στην CrispDAGComputer, προκειμένου να καταστεί δυνατός ο υπολογισμός του γράφου υπαγωγής.

7.1.3 Κλάση FuzzyDAGComputer

Η κλάση αυτή είναι η αντίστοιχη της CrispDAGComputer *για ασαφείς οντολογίες. Παρ' ότι η δομή και οι μέθοδοί της είναι αντίστοιχες, κρίνεται σκόπιμη η επανάληψη της αναφορά τους εξ' αιτίας της σημασίας της κλάσης αυτής για τη διαδικασία συλλογιστικής.*

Η κλάση FuzzyDAGComputer χρησιμοποιείται για τον υπολογισμό του συνεκτικού ακυκλικού γράφου υπαγωγής μίας $fuzzy-\mathcal{EL}^{+}$ οντολογίας, καθώς υλοποιεί τον θεωρητικό αλγόριθμο που παρουσιάζεται στην ενότητα "Υπολογισμός Άμεσων Σχέσεων Υπαγωγής" (σελ. 39). Υλοποιεί τον "Υπολογιστή Ασαφούς Γράφου Υπαγωγής", όπως αυτός παρουσιάζεται στο Σχήμα 9 (σελ. 59) της ενότητας "Υποσύστημα Υπολογισμού Γράφου Υπαγωγής".

Το UML διάγραμμα της κλάσης είναι το ακόλουθο:



Διάγραμμα 3: *Διάγραμμα UML της κλάσης FELPlusReasoner.FuzzyDAGComputer*

Υπάρχουν οι ακόλουθες αντιστοιχίσεις ανάμεσα στις διαδικασίες του αναφέρονται στον θεωρητικό αλγόριθμο και στις μεθόδους που υλοποιεί η παρούσα κλάση:

4. Η θεωρητική διαδικασία $\upsilon\pi o\lambda o\gamma\iota\sigma\mu\acute{o}\varsigma-\Sigma A\Gamma()$ υλοποιείται από τη μέθοδο computeDAG.
5. Η $\Sigma A\Gamma-\tau\alpha\xi\iota\nu\acute{o}\mu\eta\sigma\eta(A)$ του αλγορίθμου υλοποιείται από τη μέθοδο DAGClassify.
6. Η $\Sigma A\Gamma-\epsilon\iota\sigma\alpha\gamma\omega\gamma\acute{\eta}(A,\upsilon\pi o\psi\acute{\eta}\phi\iota o\iota)$ υλοποιείται από τη μέθοδο DAGInsert.

Η κλάση αυτή δέχεται το FuzzyMappingsInfoStructure input, στο οποίο αποθηκεύονται οι τα υπολογισθέντα σύνολα αντιστοιχίσεων S και R από το προηγούμενο στάδιο της διαδικασίας, ενώ χρησιμοποιεί το FuzzyDAGInfoStructure workingDataSet, το οποίο και αποτελεί

το αντικείμενο στο οποίο αποθηκεύονται τα δεδομένα του γράφου υπαγωγής επί των οποίων εργάζεται. Επίσης, ένα FuzzyDAGInfoStructure είναι σε θέση να εκτελέσει ορισμένες λειτουργίες πάνω στα δεδομένα που αποθηκεύει.

Οι υπόλοιπες μέθοδοι υλοποιούν άλλες βοηθητικές λειτουργίες, εκ των οποίων οι σημαντικότερες είναι οι εξής:

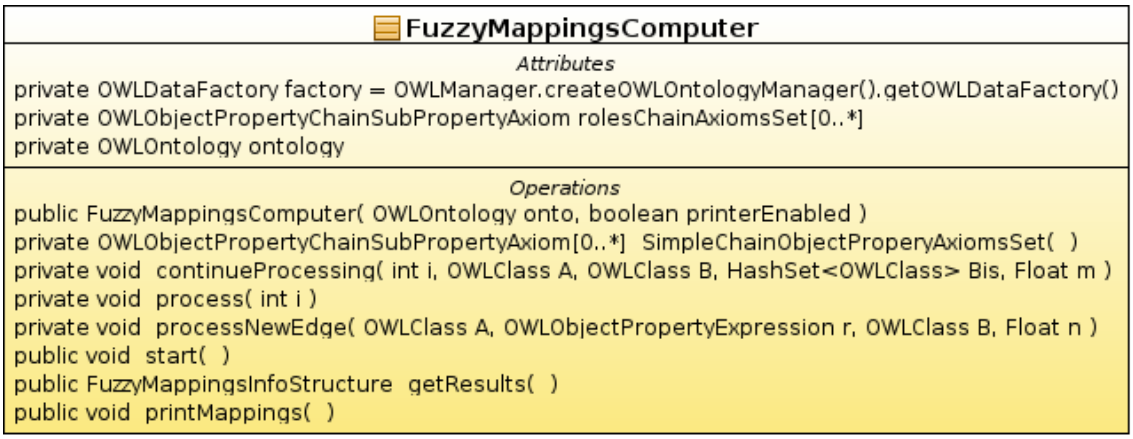
1. Η printClassification(java.net.URI outputURI) εκτυπώνει τις υπολογισθείσες πληροφορίες ταξινόμησης (κλάσεις-γονείς, ισοδύναμες κλάσεις και κλάσεις-παιδιά) για όλες τις κλάσεις μίας οντολογίας σε κάποιο αρχείο, αναφέροντας τους βαθμούς βεβαιότητας.
2. Οι getPrettyInfo(java.lang.String clsName) και getInfo(java.lang.String clsName) επιστρέφουν σε μορφή κατάλληλη για εκτύπωση πληροφορίες ταξινόμησης και βεβαιότητας για μία συγκεκριμένη κλάση.
3. Η getInferredAxioms() επιστρέφει το σύνολο των αξιωμάτων που πρέπει να εισαχθούν στην οντολογία, έτσι ώστε αυτή να περιέχει όλες τις υπολογιζόμενες πληροφορίες ταξινόμησης (υπαγωγές, ισοδυναμίες) για κάθε κλάση. Τα αξιώματα αυτά θα περιέχουν και τον βαθμό βεβαιότητάς τους ως επισημείωση (annotation), όπως περιγράφεται στην ενότητα "Απαιτήσεις Δεδομένων Εισόδου" (σελ. 45).

7.1.4 Κλάση FuzzyMappingsComputer

Η κλάση αυτή είναι η αντίστοιχη της CrispMappingsComputer *για ασαφείς οντολογίες. Παρ' ότι η δομή και οι μέθοδοί της είναι αντίστοιχες, κρίνεται σκόπιμη η επανάληψη της αναφορά τους εξ' αιτίας της σημασίας της κλάσης αυτής για τη διαδικασία συλλογιστικής.*

Η κλάση FuzzyMappingsComputer χρησιμοποιείται για τον υπολογισμό των συνόλων αντιστοιχίσεων S για κάθε ονοματική έννοια και R για κάθε ονοματικό ρόλο μίας $fuzzy - \mathcal{EL}^+$ οντολογίας. Ο αλγόριθμος που υλοποιείται αντιστοιχεί στον θεωρητικό αλγόριθμο που παρουσιάζεται στην ενότητα "Βελτιωμένος Αλγόριθμος Υπολογισμού Αντιστοιχίσεων" (σελ. 36). Υλοποιεί τον "Υπολογιστή Ασαφών Αντιστοιχίσεων", όπως αυτός παρουσιάζεται στο Σχήμα 6 (σελ. 57) της ενότητας "Υποσύστημα Υπολογισμού Συνόλων Αντιστοιχίσεων".

Το UML διάγραμμα της κλάσης αυτής παρουσιάζεται παρακάτω:



Διάγραμμα 4: *Διάγραμμα UML της κλάσης FELPlusReasoner.FuzzyMappingsComputer*

Ανάμεσα στις διαδικασίες που αναφέρονται στον θεωρητικό αλγόριθμο και στις μεθόδους που υλοποιεί η παρούσα κλάση υφίστανται οι ακόλουθες αντιστοιχίσεις:

3. Η διαδικασία $\textit{επεξεργασία}(A, X)$ υλοποιείται από τις μεθόδους process(int i) και continueProcessing(int i, org.semanticweb.owl.model.OWLClass A,

   org.semanticweb.owl.model.OWLClass B,

   java.util.HashSet<org.semanticweb.owl.model.OWLClass> Bis,

   java.lang.Float m).

   Η διαδικασία $\textit{επεξεργασία}(A, X)$ διαχωρίστηκε σε δύο μεθόδους προκειμένου να είναι ευκολότερη η επαναχρησιμοποίηση και η εποπτεία του κώδικα. Έτσι, η process καλεί την continueProcessing με αποτέλεσμα οι δύο μαζί να διεκπεραιώνουν τις ενέργειες που απαιτεί ο θεωρητικός αλγόριθμος.

4. Η διαδικασία $\textit{επεξεργασία}-\textit{νέας}-\textit{ακμής}(A, r, B, n)$ υλοποιείται από τη μέθοδο processNewEdge(org.semanticweb.owl.model.OWLClass A,

   org.semanticweb.owl.model.OWLObjectPropertyExpression r,

   org.semanticweb.owl.model.OWLClass B,

   java.lang.Float n).

Η κλάση αυτή δέχεται την κανονικοποιημένη οντολογία από το προηγούμενο στάδιο της κανονικοποίησης, η οποία και για να μπορεί να προσπελασθεί τοπικά αποθηκεύεται στη μεταβλητή org.semanticweb.owl.model.OWLOntology ontology. Επίσης, χρησιμοποιεί το αντικείμενο FuzzyMappingsInfoStructure workingDataSet προκειμένου να αποθηκεύει τα δεδομένα στα οποία εργάζεται. Τέλος, ένα FuzzyMappingsInfoStructure είναι σε θέση να εκτελέσει ορισμένες λειτουργίες πάνω στα δεδομένα που αποθηκεύει.

Άλλες σημαντικές λειτουργίες οι οποίες υλοποιούνται από μεθόδους της παρούσας κλάσης είναι οι ακόλουθες:

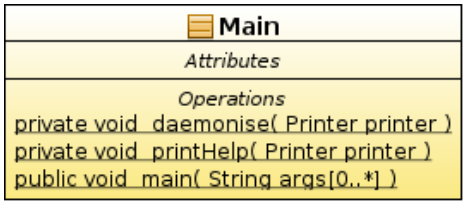
1. Πρώτον, η μέθοδος SimpleChainObjectProperyAxiomsSet(), η οποία συγκεντρώνει σε ένα σύνολο όλα τα αξιώματα τα οποία είναι αλυσίδες ρόλων με δύο στοιχεία στην υποκλάση, δηλαδή όλα τα αξιώματα μορφής $r \circ s \sqsubseteq u$, όπου r, s και u ονοματικοί ρόλοι. Στη συνέχεια επιστρέφει το σύνολο αυτό. Η μέθοδος αυτή χρησιμοποιείται από την processNewEdge.

2. Δεύτερον, η μέθοδος getResults(), η οποία επιτρέπει να λάβουμε το FuzzyMappingsInfoStructure στο οποίο αποθηκεύονται τα αποτελέσματα. Η λειτουργία αυτή χρησιμοποιείται από το κεντρικό σύστημα διαχείρισης των λειτουργιών της μηχανής, το οποίο υλοποιείται στην κλάση OntologyWorker, προκειμένου να λάβει τα υπολογισθέντα σύνολα αντιστοιχίσεων. Τα σύνολα αυτά δίδονται στην FuzzyDAGComputer, προκειμένου να καταστεί δυνατός ο υπολογισμός του γράφου υπαγωγής.

7.1.5 Κλάση Main

Η κλάση Main αποτελεί την κύρια κλάση του προγράμματος, απ' όπου και ξεκινά η εκτέλεσή του. Οι βασικές λειτουργίες που εκτελεί είναι η αναγνώριση των ορισμάτων με τα οποία καλείται το πρόγραμμα και η κατάλληλη αρχικοποίηση είτε της τοπικής, είτε της απομακρυσμένης διεπαφής ανάλογα με τις επιθυμίες του χρήστη.

Το UML διάγραμμα της κλάσης αυτής απεικονίζεται παρακάτω:



Διάγραμμα 5: *Διάγραμμα UML της κλάσης FELPlusReasoner.Main*

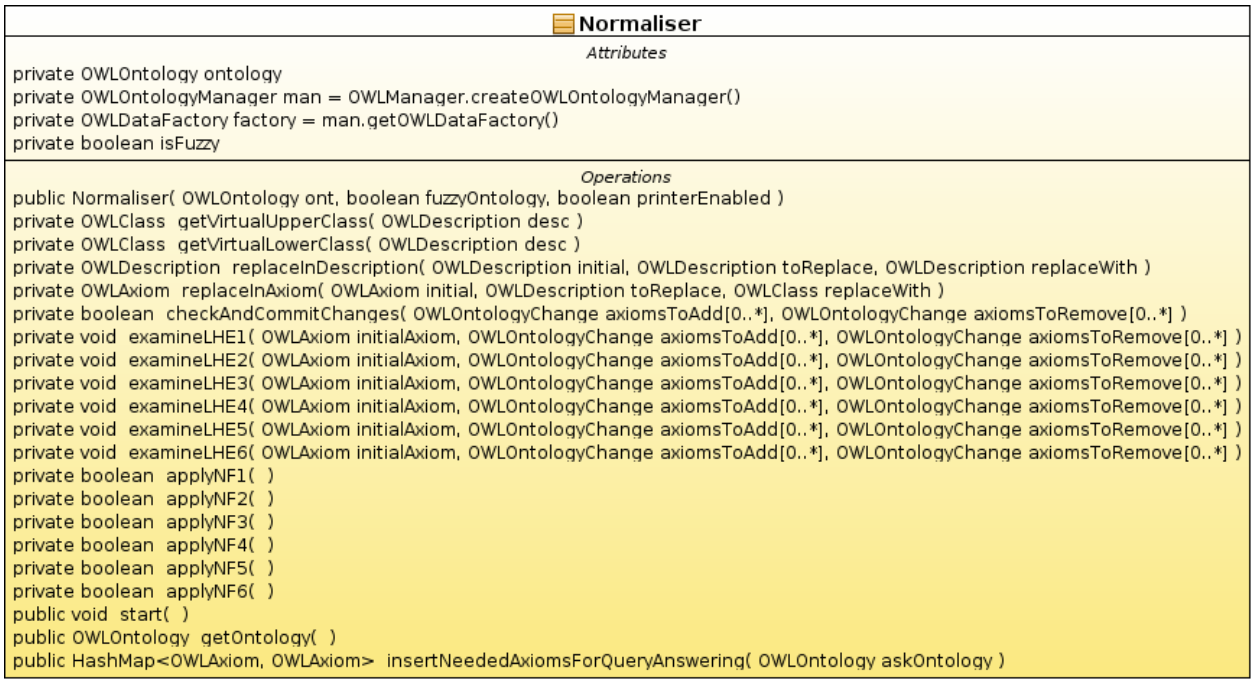
Επιγραμματικά, η συνάρτηση main δέχεται τα ορίσματα του χρήστη και είτε ενεργοποιεί άμεσα την τοπική διεπαφή με τις κατάλληλες ρυθμίσεις, είτε εκτελεί τη συνάρτηση daemonise(Printer printer), η οποία προετοιμάζει – αλλά δεν υλοποιεί – την απομακρυσμένη διεπαφή, ρυθμίζοντας το πρόγραμμα ώστε να μπορεί να δέχεται απομακρυσμένα αιτήματα.

Από την άλλη πλευρά, η συνάρτηση printHelp(Printer printer) εκτυπώνει στον χρήστη ένα σύντομο κείμενο βοηθείας το οποίο αναφέρει τα ορίσματα του προγράμματος και την ερμηνεία τους, εφ' όσον αυτός το ζητήσει ή καλέσει το πρόγραμμα χωρίς κανένα όρισμα.

7.1.6 Κλάση Normaliser

Η κλάση Normaliser χρησιμοποιείται για να κανονικοποιήσει μία $\mathcal{EL}^+$ ή $f-\mathcal{EL}^+$ οντολογία με εξαντλητική εφαρμογή 6 κανόνων σε δύο στάδια (3 κανόνες σε κάθε στάδιο), όπως ακριβώς απαιτείται και θεωρητικά. Ο αλγόριθμος που υλοποιείται για την $\mathcal{EL}^+$ αντιστοιχεί σε αυτόν που παρουσιάζεται στην ενότητα "Μετατροπή σε Κανονική Μορφή Οντολογιών EL+" (σελ. 26), ενώ ο αντίστοιχος για την $f-\mathcal{EL}^+$ παρουσιάζεται στην αντίστοιχη ενότητα (σελ. 34). Η τρέχουσα κλάση υλοποιεί το "Υποσύστημα Κανονικοποίησης Οντολογίας", όπως αυτό παρουσιάζεται στη σελ. 53, ακολουθώντας το σχετικό διάγραμμα ροής λειτουργιών (Σχήμα 5).

Το UML διάγραμμα της κλάσης αυτής παρουσιάζεται παρακάτω:



Διάγραμμα 6: *Διάγραμμα UML της κλάσης FELPlusReasoner.Normaliser*

Η λειτουργία της συγκεκριμένης κλάσης και συνεπώς η κανονικοποίηση της οντολογίας μπορεί να περιγραφεί ως εξής:

1. Αρχικά δημιουργείται ένας νέος Normaliser, ο οποίος αρχικοποιείται με την οντολογία που πρέπει να κανονικοποιήσει καθώς και με το αν αυτή είναι $\mathcal{EL}^+$ ή $f-\mathcal{EL}^+$. Αυτό γίνεται με χρήση του κατασκευαστή (constructor):

   Normaliser(org.semanticweb.owl.model.OWLOntology ont, boolean fuzzyOntology,

   boolean printerEnabled)

2. Καλείται η μέθοδος start(). Η μέθοδος αυτή ξεκινά τη διαδικασία της κανονικοποίησης ως εξής:

   i. Καλούνται οι μέθοδοι applyNF1(), applyNF2() και applyNF3() στα αξιώματα της οντολογίας. Οι μέθοδοι αυτές προσπαθούν να εφαρμόσουν τους ομώνυμους κανόνες αντικατάστασης NF1, NF2 και NF3 σε όσα από τα αξιώματα της οντολογίας είναι δυνατόν. Στο τέλος κάθε κύκλου εφαρμογής των τριών αυτών μεθόδων ελέγχεται αν συνέβησαν τροποποιήσεις στην οντολογία. Αν κατά τη διάρκεια κάποιου κύκλου εφαρμογής δεν προέκυψε καμία αντικατάσταση – πράγμα το οποίο σημαίνει πως οι κανόνες εφαρμόστηκαν εξαντλητικά – τότε δίνεται το σήμα μετάβασης στην εφαρμογή των υπολοίπων τριών κανόνων. Οι δύο αυτές λειτουργίες (έλεγχος ύπαρξης και εφαρμογή αλλαγών) υλοποιούνται από τη μέθοδο:

   checkAndCommitChanges(

   java.util.List<org.semanticweb.owl.model.OWLOntologyChange> axiomsToAdd,

   java.util.List<org.semanticweb.owl.model.OWLOntologyChange> axiomsToRemove).

Η μέθοδος αυτή δέχεται δύο λίστες με τους κανόνες που πρέπει να προστεθούν και αφαιρεθούν από την οντολογία, εφαρμόζει τις αλλαγές αυτές και επιστρέφει εάν τελικά έγιναν αλλαγές στην οντολογία ή όχι – δηλαδή αν οι λίστες δεν ήταν ή ήταν κενές αντίστοιχα.

ii. Εφ' όσον οι πρώτοι τρεις κανόνες αντικατάστασης εφαρμόστηκαν εξαντλητικά, εφαρμόζονται και οι επόμενοι τρεις κανόνες NF4, NF5 και NF6 με τρόπο ίδιο με αυτόν του προηγούμενου βήματος.

3. Η άφιξη σε αυτό το βήμα σημαίνει και το πέρας της διαδικασίας κανονικοποίησης. Το κεντρικό σύστημα διαχείρισης των λειτουργιών μπορεί να λάβει την τελική, κανονικοποιημένη, οντολογία με χρήση της μεθόδου getOntology().

Όπως είναι φανερό, οι μέθοδοι που αναφέρθηκαν δεν είναι και οι μόνες μέθοδοι της παρούσας κλάσης. Υπάρχουν και αρκετές άλλες , η λειτουργία των οποίων θα περιγραφεί επιγραμματικά παρακάτω:

1. Οι μέθοδοι examineLHE1 έως examineLHE6 είναι αυτές που υλοποιούν στην πραγματικότητα κάθε έναν από τους 6 κανόνες αντικατάστασης. Αν εξετασθεί η λειτουργία του προγράμματος σε μεγαλύτερη λεπτομέρεια, θα διαπιστωθεί ότι οι μέθοδοι applyNF1 έως applyNF6 απλά συγκεντρώνουν ένα σύνολο αξιωμάτων της οντολογίας στα οποία είναι πιθανό να μπορεί να εφαρμοστεί ο αντίστοιχος κανόνας αντικατάστασης. Στη συνέχεια, για κάθε ένα από τα αξιώματα αυτά καλούν την αντίστοιχη μέθοδο από τις examineLHE1 έως examineLHE6, η οποία και εξετάζει αν πράγματι είναι δυνατή η εφαρμογή του κανόνα και αν είναι, την υλοποιεί. Φυσικά, ούτε οι τελευταίες αυτές μέθοδοι είναι ανεξάρτητες: κάθε μία από αυτές χρησιμοποιεί άλλες μεθόδους για την αντικατάσταση των αξιωμάτων, οι οποίες περιγράφονται στη συνέχεια.

2. Η μέθοδος:

   replaceInAxiom(org.semanticweb.owl.model.OWLAxiom initial,
   org.semanticweb.owl.model.OWLDescription toReplace,
   org.semanticweb.owl.model.OWLClass replaceWith)

   αντικαθιστά σε ένα κανόνα initial της οντολογίας μία περιγραφή toReplace με την έννοια replaceWith. Η λειτουργία αυτή είναι πολύ χρήσιμη για τους κανόνες εκείνους που απαιτούν την αντικατάσταση μίας σύνθετης περιγραφής σε ένα αξίωμα της οντολογίας με μία νέα έννοια, η οποία θα υπάγει την σύνθετη αυτή περιγραφή – ή θα υπάγεται σε αυτή.

3. Η μέθοδος replaceInDescription εκτελεί τις αντίστοιχες λειτουργίες με την replaceInAxiom αλλά σε περιγραφές (OWLDescriptions) και όχι σε αξιώματα (OWLAxioms).

4. Η μέθοδος getVirtualLowerClass(org.semanticweb.owl.model.OWLDescription desc) χρησιμοποιείται από τις υπόλοιπες μεθόδους τις κλάσεις, όταν αυτές θέλουν να εισάγουν στην οντολογία μία νέα ονοματική έννοια η οποία να υπάγεται στην περιγραφή desc. Αυτό συμβαίνει γιατί παρατηρήθηκε σε αρχικό στάδιο της ανάπτυξης του συστήματος ότι ήταν σύνηθες να ζητείται η εισαγωγή πολλών νέων κλάσεων οι οποίες να υπάγονται στην ίδια περιγραφή. Όπως έδειξαν οι μετρήσεις, σε πραγματικές συνθήκες το φαινόμενο αυτό είχε ως συνέπεια την επιβάρυνση της οντολογίας με τουλάχιστον 20% περισσότερες ονοματικές κλάσεις, ενώ η επιβάρυνση στο πλήθος των αξιωμάτων ήταν ακόμα υψηλότερη. Σε άλλες οντολογίες, μπορεί τελικά αυτές οι αυτόματα εισαχθείσες

έννοιες να κατέληγαν να αποτελούν σχεδόν τις μισές ονοματικές έννοιες της οντολογίας. Η επιβράδυνση της ταχύτητας εκτέλεσης του προγράμματος ήταν φυσικά ακόμη μεγαλύτερη, καθώς ο αλγόριθμος που ακολουθείται δεν παρουσιάζει γραμμική χρονική πολυπλοκότητα ως προς το μέγεθος της εισόδου, αλλά αντιθέτως πολυωνυμική και συγκεκριμένα $O(n^4)$ ([14]).

Για την επίλυση του προβλήματος αυτού, αποφασίστηκε η υλοποίηση ενός αλγορίθμου ο οποίος θα επέτρεπε την εύρεση της κατάλληλης νέας κλάσης στην οποία θα αναφέρεται η κάθε περιγραφή που απαιτείται. Έτσι, την πρώτη φορά που ζητείται η εισαγωγή νέας έννοιας που υπάγεται σε μία περιγραφή, εισάγεται πράγματι μία νέα έννοια. Αν σε ένα επόμενο στάδιο της διαδικασίας κανονικοποίησης ο αλγόριθμος φτάνει πάλι σε ένα σημείο που έπρεπε να εισάγει μία νέα έννοια για την ίδια περιγραφή που είχε συναντήσει, τότε δε θα εισαχθεί καμία νέα έννοια, αλλά αντιθέτως επιστραφεί η έννοια η οποία είχε εισαχθεί την πρώτη φορά. Η διαδικασία δημιουργίας και ανάκτησης ζευγών περιγραφής – έννοιας θα πρέπει να είναι ταχύτατη, ώστε και να μην καθυστερεί περαιτέρω η διαδικασία της κανονικοποίησης, αλλά και ο χρόνος που "κερδίζεται" από τα επόμενα στάδια εξ' αιτίας της μείωσης του μεγέθους της οντολογίας να μην "χάνεται" λόγω της καθυστέρησης στη διαδικασία εισαγωγής νέων εννοιών.

Για την υλοποίηση της λειτουργίας αυτής χρησιμοποιείται ένα νέο αντικείμενο, το Datatypes.DACMap (Description and Class Mapper, Σύστημα Απεικόνισης Περιγραφής σε Κλάση), το οποίο είναι επιφορτισμένο με την παροχή της κατάλληλης έννοιας, κάθε φορά που ζητείται κάποια εισαγωγή για μία περιγραφή. Η λειτουργία του DACMap θα περιγραφεί στη συνέχεια. Με τον τρόπο αυτό και η εισαγωγή νέων εννοιών παραμένει γρήγορη, αλλά και η διαδικασία ταξινόμησης της οντολογίας επιταχύνθηκε σε πολύ σημαντικό βαθμό, αφού το μέγεθός της μετά την κανονικοποίηση ήταν σημαντικά μικρότερο.

5. Αντίστοιχα με την προηγούμενη μέθοδο και για την περίπτωση όπου ζητείται εισαγωγή νέας έννοιας που να υπάγει μία περιγραφή υλοποιήθηκε και η getVirtualUpperClass, η οποία χρησιμοποιεί το ίδιο DACMap με διαφορετικό προφανώς σύνολο ζευγών περιγραφής – κλάσης.
6. Η μέθοδος insertNeededAxiomsForQueryAnswering καταχωρεί στην οντολογία τα κατάλληλα αξιώματα υπαγωγής κλάσεων, ώστε να είναι δυνατή η απάντηση πολύπλοκων ερωτημάτων ισοδυναμίας ή υπαγωγής εννοιών, δηλαδή ερωτημάτων που δεν περιέχουν μόνο ονοματικές κλάσεις. Η λειτουργία της βασίζεται στο γεγονός ότι $\langle C \sqsubseteq_{\mathcal{O}} D, n\rangle$ αν και μόνο αν $\langle A \sqsubseteq_{\mathcal{O}'} B, n\rangle$, όπου $\mathcal{O}' = \mathcal{O} \cup \{\langle A \sqsubseteq C, n\rangle, \langle D \sqsubseteq B, n\rangle\}$ και A, B νέα ονόματα εννοιών.

7.1.7 Κλάση OntologyWorker

Η κλάση OntologyWorker αποτελεί κεντρική κλάση του προγράμματος, καθώς εκεί υλοποιούνται οι δύο διεπαφές και η αλληλουχία ενεργειών που απαιτείται για την ταξινόμηση της οντολογίας. Επίσης, υλοποιεί τα υποσυστήματα ελέγχου εκφραστικότητας και προεπεξεργασίας της οντολογίας, ενώ προσφέρει βασικές υπηρεσίες ελέγχου υπαγωγής και ισοδυναμίας κλάσεων, πάνω στις οποίες βασίζονται οι υλοποιούμενες διεπαφές. Συγκεντρωτικά, τα συστήματα τα οποία υλοποιούνται στην παρούσα κλάση είναι τα ακόλουθα:

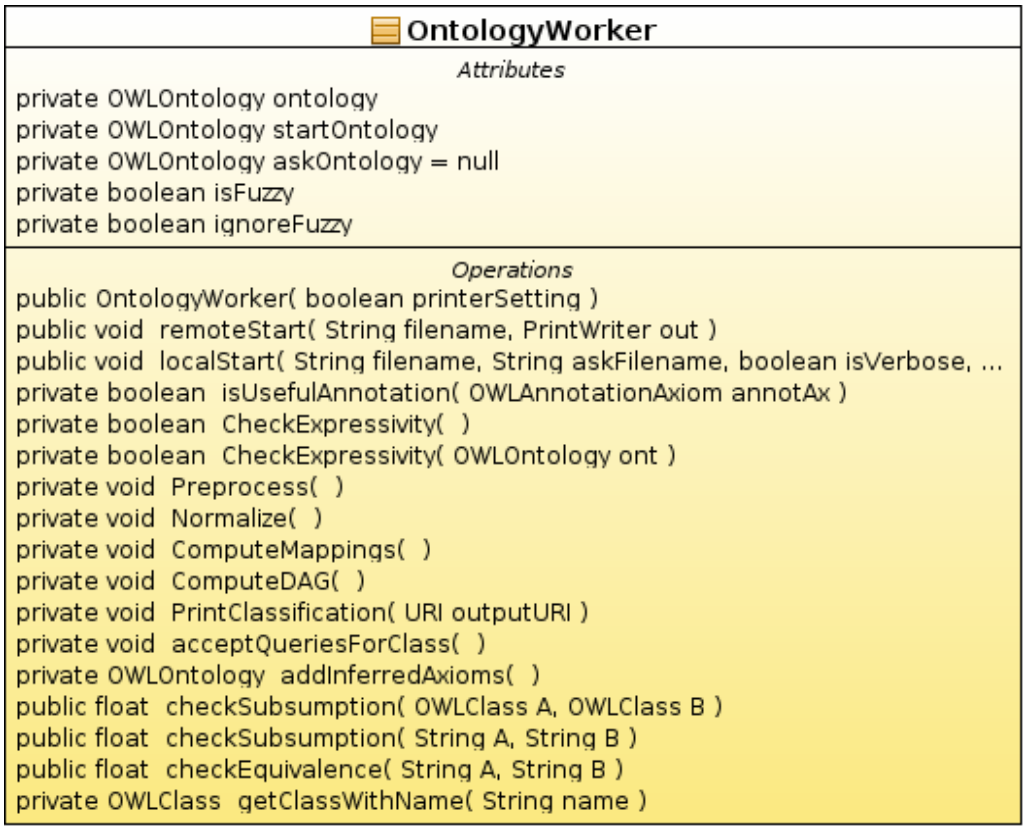
1. Κεντρικό Σύστημα Διαχείρισης Λειτουργιών (σελ. 49)

2. Υποσύστημα Ελέγχου Εκφραστικότητας Οντολογίας (σελ. 50)
3. Υποσύστημα Προεπεξεργασίας Οντολογίας (σελ. 51)
4. Μέρος του υποσυστήματος "Διεπαφή ως ανεξάρτητο πρόγραμμα" (σελ. 60)

Θα μπορούσε κάποιος να ισχυρισθεί ότι η συγκέντρωση όλων αυτών των υποσυστημάτων σε μία κλάση δεν είναι καλή σχεδιαστική επιλογή. Ότι θα έπρεπε κάθε μία από τις λειτουργίες να έχει υλοποιηθεί σε μία ξεχωριστή, κατά το δυνατόν πιο ανεξάρτητη και απλούστερη κλάση. Στην πραγματικότητα, αυτές οι αρχές κατηύθυναν τη σχεδίαση και την ανάπτυξη του προγράμματος και αυτές ακριβώς ήταν που επέβαλαν τη συγκέντρωση όλων αυτών των λειτουργιών σε μία κλάση. Θα αναφερθούν συνοπτικά οι λόγοι που οδήγησαν στη σχεδιαστική αυτή επιλογή:

1. Η λειτουργία της Διεπαφής ως ανεξάρτητο πρόγραμμα είναι άρρηκτα συνδεδεμένη με το κεντρικό σύστημα διαχείρισης λειτουργιών, κυρίως γιατί επιτρέπει πολύ περισσότερες και πολυπλοκότερες λειτουργίες σε σύγκριση με την απομακρυσμένη διεπαφή. Οι λειτουργίες αυτές απαιτούν άμεση επικοινωνία με τα υποσυστήματα της μηχανής συλλογιστικής. Αυτό έχει σαν αποτέλεσμα η απλούστερη απομακρυσμένη διεπαφή να αποτελείται σε ξεχωριστή κλάση χρησιμοποιώντας βεβαίως μεθόδους της κλάσης OntologyWorker, μιας και οι δύο διεπαφές είναι διαφορετικές όψεις του ιδίου συστήματος.
2. Οι λειτουργίες του Ελέγχου Εκφραστικότητας και της Προεπεξεργασίας είναι αρκετά απλές σε σύγκριση με τις υπόλοιπες λειτουργίες και υποσυστήματα της μηχανής συλλογιστικής, πράγμα το οποίο δεν καθιστά αναγκαία την ύπαρξη δύο νέων κλάσεων για την υλοποίησή τους, αντί για δύο μόνο μεθόδων.
3. Το Κεντρικό Σύστημα Διαχείρισης Λειτουργιών ουσιαστικά λειτουργεί καλώντας τοπικές μεθόδους της OntologyWorker.

Το διάγραμμα UML της παρούσας κλάσης απεικονίζεται παρακάτω:



Διάγραμμα 7: *UML Διάγραμμα κλάσης FELPlusReasoner.OntologyWorker*

Στη συνέχεια θα περιγραφεί επιγραμματικά η λειτουργία ορισμένων από τις σημαντικότερες μεθόδους της κλάσης OntologyWorker:

1. Η κλάση remoteStart εκτελεί τη διαδικασία ανάκτησης και ταξινόμησης μίας οντολογίας, όπως δήλωσε ότι επιθυμεί ένας χρήστης διαμέσου της απομακρυσμένης διεπαφής. Κατά τη διαδικασία της ταξινόμησης οντολογίας μέσω της απομακρυσμένης διεπαφής δεν εκτελείται το στάδιο του υπολογισμού του συνεκτικού ακυκλικού γράφου υπαγωγής (subsumption DAG). Υλοποιεί δηλαδή ένα τμήμα της απομακρυσμένης διεπαφής, ενώ το υπόλοιπο υλοποιείται στην κλάση ThrededRequestHandler.
2. Η κλάση localStart εκτελεί την ανάκτηση και την ταξινόμηση μίας οντολογίας, με βάση τις επιθυμίες του χρήστη που δόθηκαν μέσω τις τοπικής διεπαφής. Επίσης, είναι σε θέση να εκτελέσει και πιο σύνθετες λειτουργίες αν αυτό έχει ζητηθεί. Έτσι, η μέθοδος αυτή:
   i. Φορτώνει την επιθυμητή οντολογία στη μνήμη.
   ii. Αν έχει ορισθεί οντολογία που περιέχει τα ερωτήματα, τότε τη φορτώνει στη μνήμη και ελέγχει την εγκυρότητα της εκφραστικότητάς της.
   iii. Ελέγχει την εκφραστικότητα της οντολογίας με χρήση της μεθόδου CheckExpressivity().
   iv. Προεπεξεργάζεται την οντολογία με χρήση της μεθόδου Preprocess()
   v. Κανονικοποιεί την οντολογία με χρήση της Normalize().

vi. Αν έχει δηλωθεί επιθυμία εξαγωγής της κανονικοποιηθείσας οντολογίας, τότε αυτή εξάγεται στο στάδιο αυτό.

vii. Υπολογίζει τις αντιστοιχίσεις (mappings) S και R

viii. Υπολογίζει τον συνεκτικό ακυκλικό γράφο υπαγωγής, αλλά μόνο εάν κάτι τέτοιο απαιτείται για την απάντηση των αιτημάτων του χρήστη..

ix. Εκτελούνται οι λειτουργίες κατ' αρχάς της εξαγωγής πληροφοριών ταξινόμησης σε αρχείο, κατά δεύτερον της εξαγωγής της υπολογισθείσας οντολογίας που περιλαμβάνει καταγεγραμμένα τα νέα αξιώματα τα οποία προκύπτουν από τη διαδικασία συλλογιστικής και κατά τρίτον της απάντησης ερωτημάτων που ορίζονται σε εξωτερική οντολογία. Οι λειτουργίες αυτές εκτελούνται μόνο εάν έχει ζητηθεί από τον χρήστη..

x. Εκτελείται έλεγχος ισοδυναμίας και υπαγωγής για ονόματα κλάσεων που έχει θέσει ο χρήστης ως ορίσματα κατά την κλήση του συστήματος.

xi. Εφ' όσον έχει ζητηθεί, δίδεται η δυνατότητα στον χρήστη να λάβει πληροφορίες ταξινόμησης για κλάσεις τις επιλογής του, εισάγοντας τα ονόματά τους.

3. Με χρήση της μεθόδου isUsefulAnnotation ελέγχεται αν ένα αξίωμα επισήμανσης είναι χρήσιμη επισήμανση βάσει του καθορισμένου για τη $f - \mathcal{EL}^+$ προτύπου ("Απαιτήσεις Δεδομένων Εισόδου", σελ. 45).

4. Με τη μέθοδο CheckExpressivity() ελέγχεται αν η οντολογία που εισήχθη προς ταξινόμηση είναι πράγματι $\mathcal{EL}^+$ ή $f - \mathcal{EL}^+$. Η μέθοδος αυτή υλοποιεί το "Υποσύστημα Ελέγχου Εκφραστικότητας Οντολογίας" (σελ. 50). Για τη διαδικασία του ελέγχου γίνεται χρήση της κλάσης org.semanticweb.owl.util.DLExpressivityChecker του OWL API. Επίσης, στην κλάση υπάρχει και η μέθοδος CheckExpressivity(OWLOntology ont), η οποία ελέγχει την εκφραστικότητα της οντολογίας που παρέχεται ως όρισμα.

5. Η Preprocess() υλοποιεί το "Υποσύστημα Προεπεξεργασίας Οντολογίας" (σελ. 51), το οποίο και αντικαθιστά ουσιαστικά όλα τα αξιώματα ισοδυναμίας κλάσεων ή ρόλων με το ισοδύναμο ζεύγος αξιωμάτων υπαγωγής εννοιών ή ρόλων αντίστοιχα.

6. Οι μέθοδοι Normalize(), ComputeMappings() και ComputeDAG() χρησιμοποιούνται προκειμένου να εκτελέσουν τις αντίστοιχες ενέργειες πάνω στην οντολογία. Λειτουργούν καλώντας μεθόδους άλλων κλάσεων. Συγκεκριμένα:

   i. Κατά πρώτον, η Normalize() εισάγει αρχικά στην οντολογία τα αναγκαία αξιώματα που απαιτούνται για την απάντηση ερωτημάτων υπαγωγής ή ισοδυναμίας συνθέτων εννοιών, δηλαδή ερωτήματα της μορφής $C \sqsubseteq^? D$, όπου δεν ισχύει απαραίτητα ότι $C, D \in CN_{\mathcal{O}}$. Τέτοια ερωτήματα μπορούν να εισαχθούν μόνο ως αξίωμα μίας οντολογίας ερωτημάτων. Στη συνέχεια χρησιμοποιείται η κλάση Normaliser για την κανονικοποίηση της συνολικής οντολογίας.

   ii. Κατά δεύτερον, η ComputeMappings() χρησιμοποιεί, αναλόγως του είδους της οντολογίας, είτε την κλάση CrispMappingsComputer είτε την FuzzyMappingsComputer για τον υπολογισμό των αντιστοιχίσεων.

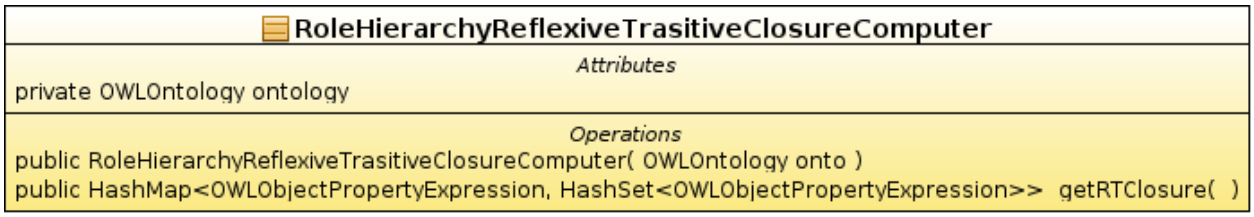
   iii. Τέλος, η ComputeDAG() χρησιμοποιεί είτε την κλάση CrispDAGComputer είτε την FuzzyDAGComputer για τον υπολογισμό του συνεκτικού ακυκλικού γράφου υπαγωγής, ανάλογα με το είδος της οντολογίας.

7. Η μέθοδος PrintClassification χρησιμοποιείται για την εξαγωγή των πληροφοριών ταινόμησης (κλάσεις-γονείς, ισοδύναμες κλάσεις και κλάσεις-παιδιά) σε κάποιο αρχείο.
8. Με τη μέθοδο acceptQueriesForClass() υλοποιείται η δυνατότητα του χρήστη να λάβει πληροφορίες για κλάσεις της επιλογής του, εισάγοντας τμήμα του ονόματός τους.
9. Η μέθοδος addInferredAxioms() προσθέτει στην οντολογία τα αξιώματα τα οποία ορίζουν την υπολογισθείσα ιεραρχία των κλάσεων και δεν περιλαμβάνονται σε αυτήν.
10. Τέλος, οι μέθοδοι checkSubsumption και checkEquivalence επιστρέφουν το βαθμό υπαγωγής ή ισοδυναμίας αντίστοιχα μεταξύ δύο κλάσεων.

7.1.8 Κλάση RoleHierarchyReflexiveTrasitiveClosureComputer

Η κλάση αυτή δίδει τη δυνατότητα υπολογισμού του ανακλαστικού – μεταβατικού κλεισίματος (reflexive – transitive closure) της ιεραρχίας ρόλων της οντολογίας, βάσει των καταγεγραμμένων σε αυτήν αξιωμάτων. Το κλείσιμο αυτό χρησιμοποιείται κατά τη διαδικασία υπολογισμού των αντιστοιχίσεων (mappings) και απαιτείται από τον θεωρητικό αλγόριθμο.

Παρακάτω δίδεται το UML διάγραμμα της συγκεκριμένης κλάσης:



Διάγραμμα 8: *Διάγραμμα UML της κλάσης FELPlusReasoner.RoleHierarchyReflexiveTrasitiveClosureComputer*

Η μέθοδος getRTClosure() είναι εκείνη η οποία υπολογίζει το ανακλαστικό – μεταβατικό κλείσιμο εφαρμόζοντας τον ορισμό του.

7.1.9 Κλάση ThreadedRequestHandler

Η κλάση ThreadedRequestHandler αναλαμβάνει την εξυπηρέτηση μίας εισερχόμενης απομακρυσμένης σύνδεσης κάποιου πελάτη στη μηχανή συλλογιστικής. Υλοποιεί ουσιαστικά την "Διεπαφή ως υπηρεσία", όπως αυτή παρουσιάζεται στη σελ. 62.

Το διάγραμμα UML της κλάσης αυτής απεικονίζεται παρακάτω:

Διάγραμμα 9: *Διάγραμμα UML της κλάσης FELPlusReasoner.ThreadedRequestHandler*

Η μέθοδος run() υλοποιεί την απομακρυσμένη διεπαφή, χρησιμοποιώντας μεθόδους της κλάσης OntologyWorker. Παρέχει δυνατότητα ταξινόμησης οντολογίας και ελέγχου βαθμού υπαγωγής ή ισοδυναμίας ονοματικών κλάσεων.

7.2 Πακέτο Datatypes

Οι κλάσεις που περιέχονται στο πακέτο Datatypes είναι οι ακόλουθες:

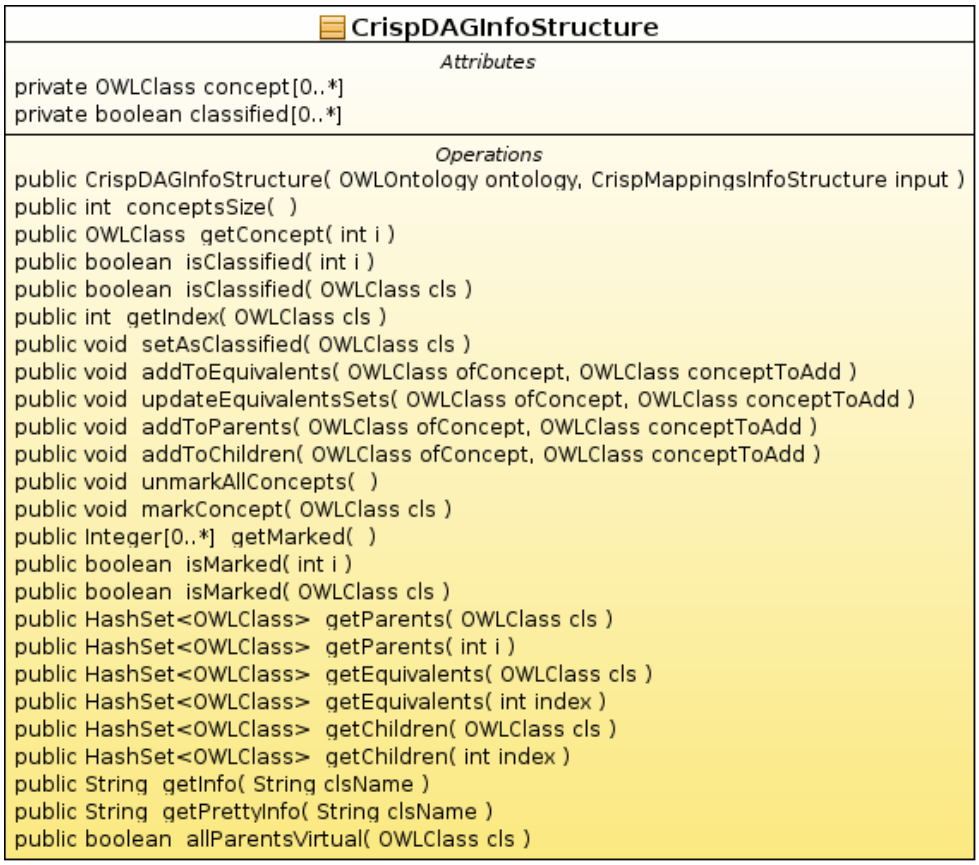
1. **CrispDAGInfoStructure:** Η κλάση CrispDAGInfoStructure αποτελεί τη δομή δεδομένων όπου αποθηκεύονται τα δεδομένα για το στάδιο του υπολογισμού του γράφου υπαγωγής (subsumption DAG) για σαφείς οντολογίες.
2. **CrispMappingsInfoStructure:** Η κλάση αυτή αποτελεί τη δομή δεδομένων όπου αποθηκεύονται τα δεδομένα για το στάδιο του υπολογισμού των αντιστοιχίσεων (mappings) S και R για σαφείς οντολογίες.
3. **CrispOWLClassesTuple:** Η κλάση CrispOWLClassesTuple υλοποιεί αντικείμενο με λειτουργία και συμπεριφορά παρόμοια με αυτή ενός 2-Tuple που αποτελείται από δύο OWLClasses.
4. **CrispQueueElement:** Τα αντικείμενα της κλάσης αυτής αποτελούν τα στοιχεία της ουράς για σαφείς οντολογίες.
5. **DACMap:** Η κλάση DACMap αποτελεί τη δομή όπου αποθηκεύονται οι αντιστοιχίσεις των νέων κλάσεων που δημιουργούνται κατά το στάδιο της κανονικοποίησης, με τις OWLDescriptions με τις οποίες σχετίζονται είτε ως υπερκλάσεις είτε ως υποκλάσεις.
6. **FuzzyDAGInfoStructure:** Η κλάση FuzzyDAGInfoStructure αποτελεί τη δομή δεδομένων όπου αποθηκεύονται τα δεδομένα για το στάδιο του υπολογισμού του γράφου υπαγωγής (subsumption DAG) για ασαφείς οντολογίες.
7. **FuzzyMappingsInfoStructure:** Η κλάση FuzzyMappingsInfoStructure αποτελεί τη δομή δεδομένων όπου αποθηκεύονται τα δεδομένα για το στάδιο του υπολογισμού των αντιστοιχίσεων (mappings) S και R για ασαφείς οντολογίες.
8. **FuzzyOWLClassesTuple:** Η κλάση FuzzyOWLClassesTuple αποτελεί τη δομή δεδομένων με λειτουργία και συμπεριφορά παρόμοια με ενός 3-Tuple.
9. **FuzzyQueueElement:** Τα αντικείμενα της κλάσης αυτής αποτελούν τα στοιχεία της ουράς για ασαφείς οντολογίες.

10. **MyOWLObjectsSet:** Η κλάση αυτή δημιουργεί ένα java.util.HashSet στο οποίο μπορούν να εισαχθούν μόνο αντικείμενα συγκεκριμένων τύπων.
11. **QueueElement:** Μία αφηρημένη κλάση που αναπαριστά ένα γενικό στοιχείο ουράς, είτε σαφούς είτε ασαφούς οντολογίας.
12. **Tuple<T,U>:** Η κλάση Tuple αποτελεί δομή δεδομένων με λειτουργία και συμπεριφορά παρόμοια με ενός 2-tuple.

Στη συνέχεια θα παρουσιαστούν περισσότερες πληροφορίες για τις κλάσεις του παρόντος πακέτου.

7.2.1 Κλάση CrispDAGInfoStructure

Η κλάση αυτή χρησιμοποιείται για την αποθήκευση των δεδομένων που απαιτούνται για τον υπολογισμό του συνεκτικού ακυκλικού γράφου υπαγωγής μίας κλασικής $\mathcal{EL}^+$ οντολογίας, καθώς και την εκτέλεση διαφόρων ενεργειών σε αυτά. Υλοποιεί τις "Δομές Δεδομένων Σαφούς Γράφου", όπως αυτές ορίζονται στην ενότητα "Υποσύστημα Υπολογισμού Γράφου Υπαγωγής" (σελ. 58).Ο υπολογισμός του γράφου υπαγωγής γίνεται από την FELPlusReasoner.CrispDAGComputer, η οποία αποθηκεύει τα δεδομένα της στην παρούσα κλάση CrispDAGInfoStructure. Στη συνέχεια παρουσιάζεται το UML διάγραμμα της CrispDAGInfoStructure:



Διάγραμμα 10: *Διάγραμμα UML της κλάσης Datatypes.CrispDAGInfoStructure*

Οι λειτουργίες που εκτελούν οι περισσότερες από αυτές τις μεθόδους γίνεται εύκολα κατανοητή τόσο από ίδιο το όνομά τους, όσο και από τη συσχέτιση αυτού με τον θεωρητικό αλγόριθμο που υλοποιεί η κλάση FELPlusReasoner.CrispDAGComputer. Υπάρχουν μέθοδοι οι οποίες σημειώνουν ("μαρκάρουν") έννοιες, ορίζουν ισοδύναμες έννοιες, έννοιες-γονείς ή έννοιες-παιδιά και πολλές άλλες. Γενικά, οι λειτουργίες της κλάσης αυτής είναι χαμηλού επιπέδου – πλην όμως κρίσιμες για την τελική ταχύτητα της υλοποίησης. Κρίνεται λοιπόν πως δεν απαιτείται ιδιαίτερη αναφορά στη λειτουργία τους, πλην της μεθόδου updateEquivalentsSets(OWLClass ofConcept, OWLClass conceptToAdd). Η μέθοδος αυτή πρέπει να καλείται μετά τη μέθοδο addToEquivalents, προκειμένου να ενημερώνει τις ήδη ισοδύναμες έννοιες των ofConcept και conceptToAdd για την εισαγωγή μίας νέας ισοδυναμίας. Αν παρατηρηθεί η λειτουργία σε περισσότερη λεπτομέρεια, θα διαπιστωθεί ότι η addToEquivalents καθιστά ισοδύναμες δύο έννοιες – και μόνον αυτές. Κάτι τέτοιο δεν είναι αρκετό, αφού όπως έχει αναφερθεί κάθε κόμβος του γράφου υπαγωγής αποτελεί μία κλάση ισοδυναμίας και δεν αποτελείται αποκλειστικά από μία έννοια. Έτσι, στη συνέχεια θα πρέπει να καλείται η updateEquivalentsSets, η οποία και ορίζει όλα τα υπολειπόμενα ζεύγη ισοδυναμίας ανάμεσα στις ισοδύναμες κλάσεις των νεο-ορισθέντος ζεύγους.

Η αναφορά περισσοτέρων λεπτομερειών γι' αυτήν ή για άλλες μεθόδους χαμηλού επιπέδου ξεφεύγει από το σκοπό μίας αναφοράς διπλωματικής εργασίας. Οι πληροφορίες αυτές πρέπει να αναζητηθούν στο Javadoc και στα άλλα σχόλια του κώδικα.

7.2.2 Κλάση CrispMappingsInfoStructure

Η παρούσα κλάση χρησιμοποιείται για την αποθήκευση των δεδομένων που απαιτούνται για τον υπολογισμό των σαφών αντιστοιχίσεων μίας $\mathcal{EL}^+$ οντολογίας, καθώς και για την εκτέλεση ενεργειών στα δεδομένα αυτά. Υλοποιεί τις "Δομές Δεδομένων Σαφών Αντιστοιχίσεων", όπως αυτές περιγράφονται στην ενότητα "Υποσύστημα Υπολογισμού Συνόλων Αντιστοιχίσεων", σελ. 55. Η κλάση αυτή χρησιμοποιείται από την FELPlusReasoner.CrispMappingsComputer.

Η κλάση CrispMappingsInfoStructure αποτελεί, μαζί με την αντίστοιχή της για ασαφείς οντολογίες FuzzyMappingsInfoStructure, ίσως την πολυπλοκότερα υλοποιηθείσα κλάση του προγράμματος. Αυτό συμβαίνει γιατί, για μία τυπική ταξινόμηση οντολογίας, ο χρόνος που δαπανάται για το στάδιο του υπολογισμού των αντιστοιχίσεων (mappings) μπορεί να είναι και περισσότερο του 75% του χρόνου που απαιτείται για να ταξινομηθεί αυτή συνολικά. Έτσι, δόθηκε πολύ μεγάλη προσοχή και δαπανήθηκε ένα μεγάλο χρονικό διάστημα στη σχεδίαση, ανάπτυξη και βελτιστοποίηση αυτής της κλάσης. Παρά το ότι η αρχική υλοποίηση ήταν σχετικά απλή και άμεσα κατανοητή, αυτή αναγκαστικά εγκαταλείφθηκε πολύ γρήγορα: έγιναν αλλαγές στον τύπο των χρησιμοποιούμενων δομών δεδομένων και εισήχθησαν νέες βοηθητικές δομές για τον περιορισμό των δυνατών εναλλακτικών περιπτώσεων που έπρεπε να διερευνά ο αλγόριθμος σε ορισμένα σημεία. Επίσης, χρησιμοποιήθηκαν δύο κρυφές μνήμες (caches) προκειμένου να παρέχουν με ταχύτητα δεδομένα τα οποία χρειάζονταν χρόνο για να επανυπολογισθούν. Όλες αυτές οι βελτιώσεις επέφεραν μία συνολική μείωση του χρόνου εκτέλεσης του αλγορίθμου κατά περισσότερο από δύο τάξεις μεγέθους.

Ανεκτίμητο εργαλείο στην προσπάθεια βελτιστοποίησης της υλοποίησης των μεθόδων αυτής της κλάσης στάθηκε ο Java Profiler που παρέχεται από το Ολοκληρωμένο Περιβάλλον Ανάπτυξης (Integrated Development Environment, IDE) NetBeans.

Αρχικά θα παρουσιαστεί το UML διάγραμμα της συγκεκριμένης κλάσης, ενώ στη συνέχεια θα εξηγηθεί η λειτουργία και η υλοποίηση συγκεκριμένων μεθόδων, οι οποίες είτε υλοποιούνται με μία πρωτότυπη ιδέα που επιταχύνει σημαντικά την ταχύτητα εκτέλεσης του προγράμματος, είτε χρησιμοποιούν μία πολύπλοκη δομή αναπαράστασης στη μνήμη. Σε κάθε περίπτωση, περισσότερες λεπτομέρειες μπορούν να βρεθούν στα σχόλια του κώδικα και στο Javadoc.

Διάγραμμα 11: *Διάγραμμα UML της κλάσης Datatypes.CrispMappingsInfoStructure*

Κατ' αρχάς, για την αναφορά σε κάποια έννοια ή κάποιον ρόλο είναι πολλές φορές ταχύτερο να μη χρησιμοποιείται η ίδια ή έννοια ή ο ρόλος, αλλά κάποιος δείκτης σε πίνακα που τους περιέχει. Αρχικές υλοποιήσεις για εύρεση του δείκτη κάποιου αντικειμένου με δυαδική αναζήτηση δεν ήταν ικανοποιητικά αποδοτικές και κλιμακώσιμες (scalable), οπότε χρησιμοποιήθηκαν αντικείμενα τύπου java.util.HashMap για την αντιστοίχηση ζευγών αντικειμένων – τιμών. Τα αντικείμενα αυτά αντιστοιχούν ένα αντικείμενο-κλειδί με ένα αντικείμενο-τιμή και επιτρέπουν την προσπέλαση των πληροφοριών σε χρόνο σημαντικά μικρότερο από τις αρχικές υλοποιήσεις.

Επίσης, τα σύνολα R για κάθε ονοματικό ρόλο (role sets) δεν αποθηκεύονται στη μνήμη απ' ευθείας. Μία απλή υλοποίηση θα ήταν η χρήση του αντικειμένου CrispOWLClassesTuple, το οποίο θα επέτρεπε τη δημιουργία ενός ζεύγους κλάσεων και τη μετέπειτα αποθήκευσή του σε ένα σύνολο, όπως μας οδηγεί η λογική και μία πρώτη θεώρηση του κώδικα. Στην πραγματικότητα όμως, κάτι τέτοιο δεν είναι αποδοτικό για δύο λόγους:

1. Μια τέτοια μορφή αναπαράστασης περιέχει πολλή πλεονάζουσα πληροφορία, καθώς μπορεί σε ένα σύνολο να υπάρχουν πολλές CrispOWLClassesTuples των οποίων οι

κλάσεις που βρίσκονται στην πρώτη ή στη δεύτερη θέση να ταυτίζονται. Κάτι τέτοιο δε φαίνεται εκ πρώτης όψεως σημαντικό, αλλά όμως η πλεονάζουσα αυτή πληροφορία είναι η αιτία σημαντικών προβλημάτων χρήσης μνήμης σε μεγάλες οντολογίες. Εκτός αυτού, η ίδια η προσπέλαση της μνήμης και οι διαδικασίες συλλογής απορριμάτων (garbage collection) της Java είναι πολύ βραδύτερες. Το πρόβλημα αυτό σε συνδυασμό με τα προβλήματα που εμφανίζονται όταν μία εφαρμογή καταναλώνει σχεδόν όλη τη φυσική μνήμη του συστήματος (swapping και thrashing), είχαν ως αποτέλεσμα τον πολλαπλάσιο χρόνο εκτέλεσης ή ακόμα και αδυναμία ταξινόμησης της οντολογίας σε αποδεκτό χρόνο.

Το πρόβλημα αυτό μπορεί να λυθεί αν υπήρχε κάποιος τρόπος ομαδοποίησης των ζευγών κλάσεων και αναπαράστασής τους με κάποια διαφορετική μορφή, που θα επέτρεπε μικρότερη χρήση μνήμης.

2. Υπάρχουν στάδια του αλγορίθμου τα οποία απαιτούν εκτέλεση ενεργειών για όλα τα ζεύγη κλάσεων κάποιου συνόλου R ρόλων, των οποίων η δεύτερη κλάση ταυτίζεται με κάποια συγκεκριμένη. Το αν ένα ζεύγος κλάσεων είναι έγκυρο και πρέπει συνεπώς να εκτελεσθεί κάποια ενέργεια γι' αυτό απαιτεί εν γένει την ένα – προς – ένα εξέταση των CrispOWLClassesTuples του συνόλου. Επιπλέον, ο αλγόριθμος οφείλει να εκτελέσει πολλές φορές μία τέτοια λειτουργία κατά τη διάρκεια ταξινόμησης της οντολογίας, με αποτέλεσμα την σημαντικότατη καθυστέρηση της λειτουργίας του.

   Είναι λοιπόν αναγκαίο να βρεθεί ένας τρόπος αποθήκευσης και ταχύτατης ανάκτησης των Tuples με βάση την κλάση που αυτά αναφέρουν στη δεύτερη θέση τους, ώστε να περιορισθεί ο χώρος αναζήτησης και να επιταχυνθεί η λειτουργία του αλγορίθμου.

Αρχικά λοιπόν και εξ' αιτίας του παρακάτω προβλήματος σημαντικής χρήσης μνήμης και επιβράδυνσης του αλγορίθμου λόγω της απλής αναπαράστασης των συνόλων, αποφασίστηκε η χρήση μίας πολυπλοκότερης μεθόδου αναπαράστασης, η οποία όχι μόνο έλυσε αυτό το πρόβλημα, αλλά επιτάχυνε και σημαντικά την εκτέλεση του αλγορίθμου. Αποφασίστηκε η χρήση για κάθε ονοματικό ρόλο ενός HashMap<OWLClass, HashSet<OWLClass>> για την αναπαράσταση του συνόλου R αυτού: Το HashMap αυτό περιέχει ως στοιχείο-κλειδί την κλάση που βρίσκεται στη δεύτερη θέση ενός Tuple, ενώ το στοιχείο-τιμή θα ήταν ένα σύνολο που θα περιείχε τις κλάσεις που θα βρίσκονταν στην πρώτη θέση εκείνων των Tuples των οποίων η δεύτερη θέση περιέχει την κλάση-κλειδί. Έγινε ουσιαστικά μία ομαδοποίηση των Tuples κάθε συνόλου R με βάση το δεύτερό τους στοιχείο. Η ανάκτηση κατ' αυτόν τον τρόπο γίνεται με χρήση της μεθόδου getRoleSetSubsetByTuple1.

Ταυτόχρονα όμως, άλλα σημεία του αλγορίθμου απαιτούν τη χρήση ή υλοποιούνται απλούστερα αν χρησιμοποιηθεί η "παραδοσιακή" – απλή αναπαράσταση των Tuples, δηλαδή ως ζεύγη – στοιχεία ενός συνόλου για κάθε ονοματικό ρόλο. Έχουμε αναφέρει πως μία τέτοια αναπαράσταση έχει ήδη κριθεί ανέφικτη λόγω προβλημάτων χρήσης μνήμης κατά την ταξινόμηση μεγάλων οντολογιών, οπότε ανέφικτη είναι και η αποθήκευση των δύο μεθόδων αναπαράστασης ταυτόχρονα στη μνήμη. Από την άλλη πλευρά, η κατασκευή κάθε φορά της "παραδοσιακής" αναπαράσταση χρειάζεται σημαντικό χρόνο και έπρεπε να επαναλαμβάνεται πολλές φορές κατά τη διάρκεια ταξινόμησης της οντολογίας. Για την επίλυση αυτού του προβλήματος χρησιμοποιείται η λύση της χρήσης ενός αντικειμένου αναπαράστασης μίας κρυφής μνήμης (cache memory). Η μνήμη αυτή αποθηκεύει τις "παραδοσιακές" αναπαραστάσεις των συχνότερα ζητούμενων συνόλων ρόλων (role sets). Έτσι, και οι απαιτήσεις μνήμης παραμένουν ελεγχόμενες, αφού η κρυφή μνήμη καταλαμβάνει πρακτικά σταθερό μέγεθος, αλλά και σπάνια χρειάζεται κάποιο σύνολο R σε παραδοσιακή αναπαράσταση το οποίο να μην υπάρχει στην κρυφή μνήμη. Όλα αυτά έχουν ως αποτέλεσμα να γίνεται η λειτουργία της ταξινόμησης ακόμα ταχύτερα. Το

αντικείμενο κρυφής μνήμης που χρησιμοποιείται έχει υλοποιηθεί στην ελεύθερη βιβλιοθήκη Whirly Cache. Η ανάκτηση της "παραδοσιακής" αναπαράστασης γίνεται με χρήση της μεθόδου getRoleSet.

Οι αποθηκευμένες αναπαραστάσεις των σύνολα ρόλων στην κρυφή μνήμη πρέπει φυσικά να ενημερώνονται να κάθε φορά που συμβαίνει μία μεταβολή στην κύρια αναπαράσταση, και αυτό γιατί η ακύρωση (invalidation) του συνόλου από την κρυφή μνήμη και ο επανυπολογισμός του παρουσιάζει πολύ μεγαλύτερο κόστος από αυτό της μικρής τροποποίησης του αποθηκευμένου αντικειμένου, όποτε μία αλλαγή απαιτεί. Όλες αυτές οι λειτουργίες εισαγωγής και ενημέρωσης υλοποιούνται από τη μέθοδο addTupleToRoleSet.

Όπως έχει ήδη αναφερθεί, συχνά κατά την εκτέλεση του αλγορίθμου απαιτείται να βρεθούν τα Tuples όλων των ρόλων, τα οποία έχουν στη δεύτερη θέση τους κάποια συγκεκριμένη κλάση. Οι προηγούμενες τεχνικές βελτιστοποίησης επιτρέπουν την γρήγορη εκτέλεση μίας τέτοιας αναζήτησης, εφ' όσον όμως γνωρίζουμε τον ρόλο. Αυτό όμως δε συμβαίνει, και έτσι μία απλή υλοποίηση του αλγορίθμου απαιτείται να ελέγχει όλους τους ρόλους της οντολογίας για ύπαρξη των σχετικών Tuples στο σύνολο R (role set) κάθε ενός. Από υλοποιήσεις και μετρήσεις έγινε αντιληπτό πως η ταχύτητα μίας τέτοιας υλοποίησης είναι πολύ περιορισμένη, καθώς εξετάζονται πολλοί περισσότεροι ρόλοι από όσους πραγματικά ενδιαφέρουν.

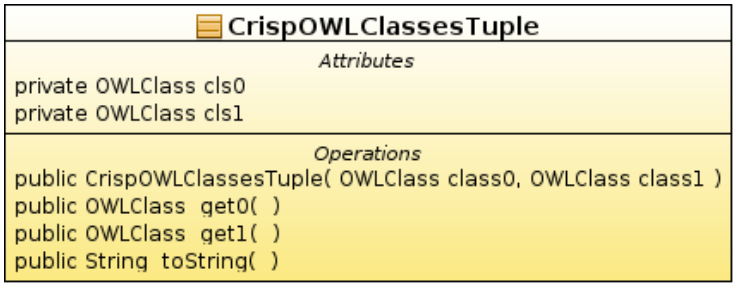
Προκειμένου λοιπόν να υπάρξει και ένας επιπλέον περιορισμός του χώρου αναζήτησης χρησιμοποιείται μία δομή δεδομένων (HashMap), στην οποία αποθηκεύεται ποιων ρόλων το role set περιέχει Tuples στων οποίων τη δεύτερη θέση βρίσκεται μία συγκεκριμένη κλάση. Η επιτάχυνση της εκτέλεσης του αλγορίθμου μετά την εισαγωγή αυτής της βοηθητικής δομής οφείλεται στο γεγονός ότι στα κρίσιμα σημεία που καθυστερούν το αλγόριθμο, γνωρίζουμε ποια ακριβώς είναι αυτή η κλάση που βρίσκεται στη δεύτερη θέση: μένει να βρούμε ποιοι ρόλοι περιέχουν τα σχετικά Tuples και ποιες είναι οι έννοιες της πρώτης θέσης. Το πρώτο αυτό στάδιο της εύρεσης των σχετικών ρόλων υποστηρίζεται από αυτή τη δομή. Συγκεκριμένα, για κάθε προσθήκη ενός νέου Tuple σε κάποιο σύνολο R (role set) γίνεται μία εγγραφή στη δομή αυτή, η οποία ορίζει πως η κλάση της δεύτερης θέσης σχετίζεται και με τον τρέχοντα ρόλο (μέθοδος addTupleToRoleSet). Όταν αργότερα πρέπει να ανακτηθούν όλοι αυτοί οι σχετιζόμενοι ρόλοι, αρκεί να προσπελασθεί η τιμή του HashMap χρησιμοποιώντας ως τιμή-κλειδί την κλάση της δεύτερης θέσης. Το αντικείμενο που επιστρέφεται είναι ένα σύνολο το οποίο περιέχει μόνο τους επιθυμητούς ρόλους και οι οποίοι συνήθως είναι ορισμένες τάξεις μεγέθους λιγότεροι σε σχέση με το σύνολο των ονοματικών ρόλων.

Τέλος, ένα άλλο σημείο στο οποίο έγινε σημαντική βελτιστοποίηση της υλοποίησης του αλγορίθμου είναι στο σημείο της εύρεσης των αναγκαίων εγγραφών που πρέπει να εισαχθούν σε κάποια ουρά, αναλόγως της εγγραφής που εισάγεται σε αυτή. Όπως εύκολα παρατηρείται και από τον θεωρητικό αλγόριθμο, το σύνολο των εγγραφών που αντιστοιχούν στην εισαγωγή κάποιας περιγραφής εξαρτάται μόνο από την περιγραφή αυτή και την αρχική οντολογία, και όχι από δεδομένα που μεταβάλλονται κατά τη διάρκεια της ταξινόμησης. Προκειμένου να επιταχυνθεί περαιτέρω η εκτέλεση του αλγορίθμου, τα ζεύγη περιγραφών και συνόλου αναγκαίων εγγραφών αποθηκεύονται κατά τον πρώτο υπολογισμό τους σε μία δεύτερη κρυφή μνήμη. Αυτό καθιστά τις μετέπειτα πολυάριθμες αιτήσεις επανυπολογισμού τους ταχύτατες, καθώς απλά ανακτώνται τα αποθηκευμένα δεδομένα της κρυφής μνήμης και δεν λαμβάνει χώρα νέος αργός επανυπολογισμός. Η διαδικασία προσθήκης εγγραφών σε ουρά υλοποιείται από τη μέθοδο addNecessaryToQueue, η οποία προσθέτει στην ουρά τις εγγραφές που ορίζει η μέθοδος neededQueueEntries. Η τελευταία αυτή μέθοδος είναι εκείνη που παρέχει το σύνολο των αναγκαίων εγγραφών για κάποια περιγραφή ελέγχοντας πρώτα τις καταχωρίσεις της κρυφής μνήμης.

Αν δεν υφίσταται σχετική καταχώριση, τότε υπολογίζει το σύνολο των απαιτούμενων εγγραφών για την περιγραφή και το αποθηκεύει στην κρυφή μνήμη. Με τον τρόπο αυτό επιταχύνεται σημαντικά η εκτέλεση του υπολογισμού των αντιστοιχίσεων για κάθε έννοια της οντολογίας.

7.2.3 Κλάση CrispOWLClassesTuple

Η κλάση CrispOWLClassesTuple χρησιμοποιείται για την αναπαράσταση των 2-Tuples που χρησιμοποιούνται ως στοιχεία του συνόλου R κάποιου ονοματικού ρόλου μίας $\mathcal{EL}^+$ οντολογίας. Όπως αναφέρεται και στη θεωρητική περιγραφή του αλγορίθμου, τα στοιχεία των συνόλων αυτών είναι ουσιαστικά ζεύγη εννοιών, μορφής (A, B). Για την αναπαράστασή κάθε ζεύγους υλοποιήθηκε λοιπόν η παρούσα κλάση, της οποίας το διάγραμμα UML είναι αυτό που απεικονίζεται παρακάτω:

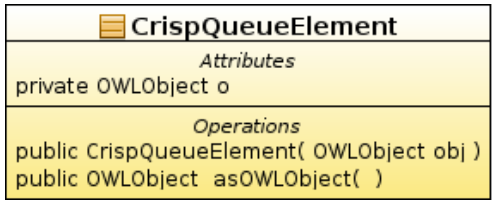


Διάγραμμα 12: *Διάγραμμα UML κλάσης Datatypes.CrispOWLClassesTuple*

Όπως γίνεται αντιληπτό, η λειτουργία της παρούσας κλάσης είναι απλή: καθιστά δυνατή τη δημιουργία ενός αντικειμένου που θα περιέχει ένα ζεύγος κλάσεων, καθώς και τη μετέπειτα ανάκτηση κάθε κλάσης του ζεύγους με τις get0() και get1().

7.2.4 Κλάση CrispQueueElement

Η κλάση CrispQueueElement επεκτείνει την QueueElement και χρησιμοποιείται για την αποθήκευση των στοιχείων ουράς που απαιτούνται για τον υπολογισμό των αντιστοιχίσεων σε σαφείς $\mathcal{EL}^+$ οντολογίες. Κάθε στοιχείο της ουράς περιέχει ουσιαστικά ένα OWLObject. Η υλοποίησή της είναι απλή, όπως διακρίνεται και από το UML διάγραμμά της:



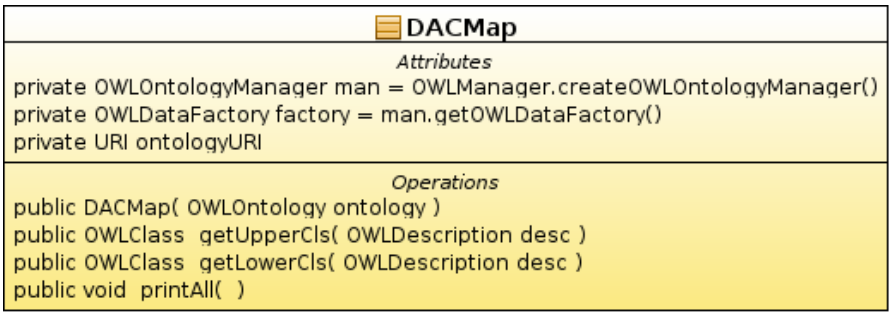
Διάγραμμα 13: *Διάγραμμα UML της κλάσης Datatypes.CrispQueueElement*

Η κλάση αυτή έχει μόνο δύο μεθόδους: τη μέθοδο-κατασκευαστή που δημιουργεί ένα νέο στοιχείο ουράς και τη μέθοδο που ανακτά το αντικείμενο που περιλαμβάνει το στοιχείο αυτό.

7.2.5 Κλάση DACMap

Η λειτουργία της κλάσης αυτής έχει ήδη αναφερθεί στην ενότητα "Κλάση Normaliser" (σελ. 70). Βασική αρμοδιότητά της λοιπόν είναι να δίδει στην κλάση Normaliser το κατάλληλο όνομα έννοιας, κάθε φορά που αυτή ζητά την εισαγωγή μίας νέας έννοιας κατά τη διάρκεια της κανονικοποίησης της οντολογίας. Έχει παρατηρηθεί ότι παρουσιάζονται πολλαπλές αυτόματα εισαχθείσες ισοδύναμες έννοιες σε μία κανονικοποιημένη οντολογία, επιβραδύνοντας δυσανάλογα την ταχύτητα ταξινόμησής της. Για τον περιορισμό αυτού του φαινομένου, η κλάση αυτή αποθηκεύει ποια περιγραφή από αυτές για τις οποίες έχει ζητηθεί εισαγωγή νέας σχετιζόμενης κλάσης σχετίζεται με ποια έννοια και κατά ποιον τρόπο (ως υποκλάση ή ως υπερκλάση). Έτσι, είναι δυνατή η εισαγωγή μίας μόνο έννοιας που θα σχετίζεται με κάποια περιγραφή, ελαχιστοποιώντας το μέγεθος της κανονικοποιημένης οντολογίας και επιταχύνοντας τον αλγόριθμο.

Το διάγραμμα UML της παρούσας κλάσης παρουσιάζεται παρακάτω:



Διάγραμμα 14: *Διάγραμμα UML της κλάσης Datatypes.DACMap*

Οι δύο κύριες μέθοδοι της κλάσης είναι:

1. Πρώτον, η μέθοδος getLowerCls(OWLDescription desc), η οποία επιστρέφει την κατάλληλη νέα ή ήδη εισαχθείσα κλάση που είναι υποκλάση της περιγραφής που ζητείται.
2. Δεύτερον, η μέθοδος getUpperCls(OWLDescription desc), η οποία και επιστρέφει την κατάλληλη νέα ή ήδη εισαχθείσα κλάση που υπάγει την περιγραφή που ζητείται.

7.2.6 Κλάση FuzzyDAGInfoStructure

Η κλάση αυτή χρησιμοποιείται για την αποθήκευση των δεδομένων που απαιτούνται για τον υπολογισμό του συνεκτικού ακυκλικού γράφου υπαγωγής μίας $f-\mathcal{EL}^{+}$ οντολογίας, καθώς και την εκτέλεση διαφόρων ενεργειών σε αυτά. Υλοποιεί τις "Δομές Δεδομένων Ασαφούς Γράφου", όπως αυτές ορίζονται στην ενότητα "Υποσύστημα Υπολογισμού Γράφου Υπαγωγής" (σελ. 58). Ο υπολογισμός του γράφου υπαγωγής γίνεται από την FELPlusReasoner.FuzzyDAGComputer, η οποία αποθηκεύει τα δεδομένα της στην παρούσα κλάση FuzzyDAGInfoStructure. Εν συνεχεία παρουσιάζεται το UML διάγραμμα της FuzzyDAGInfoStructure:

```
FuzzyDAGInfoStructure
Attributes
private OWLClass concept[0..*]
private boolean classified[0..*]
Operations
public FuzzyDAGInfoStructure( OWLOntology ontology, FuzzyMappingsInfoStructure input )
public int  conceptsSize(  )
public OWLClass  getConcept( int i )
public boolean  isClassified( int i )
public boolean  isClassified( OWLClass cls )
public int  getIndex( OWLClass cls )
public void  setAsClassified( OWLClass cls )
public void  addToEquivalents( OWLClass ofConcept, OWLClass conceptToAdd, Float n )
public void  updateEquivalentsSets( OWLClass ofConcept, OWLClass conceptToAdd, Float n )
public void  addToParents( OWLClass ofConcept, OWLClass conceptToAdd, Float n )
public void  addToChildren( OWLClass ofConcept, OWLClass conceptToAdd, Float n )
public void  unmarkAllConcepts(  )
public void  markConcept( OWLClass cls )
public Integer[0..*]  getMarked(  )
public boolean  isMarked( int i )
public boolean  isMarked( OWLClass cls )
public OWLClass[0..*]  getParents( OWLClass cls )
public OWLClass[0..*]  getParents( int i )
public OWLClass[0..*]  getEquivalents( OWLClass cls )
public OWLClass[0..*]  getEquivalents( int index )
public OWLClass[0..*]  getChildren( int i )
public OWLClass[0..*]  getChildren( OWLClass cls )
public String  getInfo( String clsName )
public String  getPrettyInfo( String clsName )
public boolean  allParentsVirtual( OWLClass cls )
public HashMap<OWLClass, Float>  getEquivalentsMap( OWLClass cls )
public HashMap<OWLClass, Float>  getChildrenMap( OWLClass cls )
```

Διάγραμμα 15: *Διάγραμμα UML της κλάσης Datatypes.FuzzyDAGInfoStructure*

Οι λειτουργίες που εκτελούν οι περισσότερες από τις μεθόδους αυτές γίνονται εύκολα κατανοητές τόσο από ίδιο το όνομά τους, όσο και από τη συσχέτιση αυτών με τον θεωρητικό αλγόριθμο που υλοποιεί η κλάση FELPlusReasoner.FuzzyDAGComputer. Υπάρχουν μέθοδοι οι οποίες σημειώνουν ("μαρκάρουν") έννοιες, ορίζουν ισοδύναμες έννοιες, έννοιες-γονείς ή έννοιες-παιδιά και πολλές άλλες. Γενικά, οι λειτουργίες της κλάσης αυτής είναι χαμηλού επιπέδου – πλην όμως κρίσιμου για την τελική ταχύτητα της υλοποίησης –. Κρίνεται λοιπόν πως δεν απαιτείται ιδιαίτερη αναφορά στη λειτουργία τους, πλην της μεθόδου updateEquivalentsSets(OWLClass ofConcept, OWLClass conceptToAdd, Float n). Η λειτουργία της μεθόδου αυτής είναι αντίστοιχη με εκείνη που περιγράφεται για την ομώνυμη μέθοδο της κλάσης CrispDAGInfoStructure, η οποία και περιγράφεται στην ενότητα "Κλάση CrispDAGInfoStructure" (σελ. 79), αν λάβουμε υπ' όψιν την ύπαρξη βαθμών βεβαιότητας.

7.2.7 Κλάση FuzzyMappingsInfoStructure

Η παρούσα κλάση χρησιμοποιείται για την αποθήκευση των δεδομένων που απαιτούνται για τον υπολογισμό των ασαφών αντιστοιχίσεων μίας $f-\mathcal{EL}^{+}$ οντολογίας, καθώς και για την εκτέλεση ενεργειών στα δεδομένα αυτά. Υλοποιεί τις "Δομές Δεδομένων Ασαφών Αντιστοιχίσε-

ων", όπως αυτές περιγράφονται στην ενότητα "Υποσύστημα Υπολογισμού Συνόλων Αντιστοιχίσεων" (σελ. 55). Η κλάση αυτή χρησιμοποιείται από την FELPlusReasoner.FuzzyMappingsComputer.

Για τους ιδίους λόγους όπως και η κλάση CrispMappingsInfoStructure, αποτελεί η FuzzyMappingsInfoStructure ίσως την πολυπλοκότερα υλοποιηθείσα κλάση του προγράμματος. Οι λόγοι για τους οποίους συμβαίνει αυτό είναι ακριβώς οι ίδιοι με αυτούς που αναλύονται στην ενότητα "Κλάση CrispMappingsInfoStructure" (σελ. 81) και δεν κρίνεται σκόπιμο να επαναληφθεί η παρουσίασή τους. Το ίδιο συμβαίνει και με τις λύσεις που δόθηκαν για την επίλυση των προβλημάτων και την επιτάχυνση του αλγορίθμου. Συνεπώς, θα παρουσιαστεί μόνο το UML διάγραμμα της συγκεκριμένης κλάσης και ορισμένες μόνο μέθοδοι οι οποίες παρουσιάζουν διαφορές άξιες λόγου. Αν ο αναγνώστης επιθυμεί περισσότερες πληροφορίες για την υλοποίηση της παρούσας κλάσης παρακαλείται να ανατρέξει στην προαναφερθείσα ενότητα για την κλάση CrispMappingsInfoStructure καθώς και στο Javadoc.

Η βασική αιτία η οποία οδήγησε σε διαφορές της υλοποίησης αυτής της κλάσης μεγαλύτερες από τις παρατηρούμενες στα υπόλοιπα στάδια του αλγορίθμου, ήταν η δεύτερη από τις διαφορές που περιγράφεται στην ενότητα "Αλγόριθμος Υπολογισμού Αντιστοιχίσεων", σελ. 43. Συγκεκριμένα, ήταν η ανάγκη ο αλγόριθμος να εισάγει τους ορθούς βαθμούς στα σύνολα απαιτούμενων εγγραφών ουράς για περιγραφές τύπου $\exists r.A$, δηλαδή για σύνολα απαιτούμενων εγγραφών ουρών μορφής $\hat{\mathcal{O}}(\exists r.A)$. Η διαφορά έγκειται στο γεγονός ότι ο βαθμός βεβαιότητας κάθε εγγραφής ουράς δε θα πρέπει να υπερβαίνει το βαθμό βεβαιότητας ισχύος του αιτίου που προκαλεί την εισαγωγή του. Ο μέγιστος αυτός βαθμός βεβαιότητας του αιτίου υπολογίζεται από την FELPlusReasoner.FuzzyMappingsComputer, ενώ από την πλευρά της παρούσας κλάσης FuzzyMappingsInfoStructure απαιτήθηκαν τροποποιήσεις σε σχέση με τον σαφή αλγόριθμο στη μέθοδο neededQueueEntries, καθώς και εισαγωγή της νέας μεθόδου addNecessaryToQueueWithMaxN. Οι αλλαγές συνοψίζονται στα εξής:

1. Η μέθοδος neededQueueEntries δεν επιστρέφει πλέον απλά το σύνολο των εγγραφών, αλλά επιστρέφει ένα αντικείμενο το οποίο περιέχει το σύνολο αυτό και έναν αριθμό, ο οποίος αντιστοιχεί στο μέγιστο βαθμό βεβαιότητας που απαντάται στις εγγραφές του συνόλου. Το περιεχόμενο αυτό του μεγίστου βαθμού βεβαιότητας είναι έγκυρο μόνο για σύνολα που αντιστοιχούν σε περιγραφές τύπου $\exists r.A$. Ο συνδυασμός αυτός συνόλου εγγραφών και μεγίστου βαθμού αποθηκεύεται στην κρυφή μνήμη NQECache όπως και στην περίπτωση του κλασικού (σαφούς) αλγορίθμου.

2. Στη συνέχεια αναλαμβάνει τον έλεγχο η μέθοδος addNecessaryToQueue ή η μέθοδος addNecessaryToQueueWithMaxN. Στην πρώτη περίπτωση η εισαγωγή των στοιχείων στην ουρά γίνεται κανονικά, αγνοώντας τον μέγιστο βαθμό βεβαιότητας. Η δεύτερη μέθοδος addNecessaryToQueueWithMaxN χρησιμεύει όποτε ο αλγόριθμος πρέπει να προσθέσει το σύνολο $\hat{\mathcal{O}}(\exists r.A)$ των απαιτούμενων εγγραφών ουράς σε μία ουρά. Γνωρίζοντας τον μέγιστο βαθμό βεβαιότητας των εγγραφών του συνόλου, ο οποίος μάλιστα αποθηκεύεται και σε κρυφή μνήμη, και τον μέγιστο επιτρεπτό βαθμό βεβαιότητας εξ' αιτίας του κανόνα συμπλήρωσης R3, είναι δυνατή η αποφυγή της δαπανηρής λειτουργίας ελέγχου του συνόλου των εγγραφών για τις περιπτώσεις όπου ο μέγιστος εμφανιζόμενος βαθμός είναι μικρότερος ή ίσος του μεγίστου επιτρεπτού. Η μέθοδος αυτή επιτάχυνσης βοηθά κατά πολύ σημαντικό βαθμό στην αύξηση της ταχύτητας ταξινόμησης οντολογιών, καθώς επιτρέπει την σπανιότερη εκτέλεση της χρονικά δαπανηρής λειτουργίας του ελέγχου, εγγραφή προς εγγραφή, κάθε στοιχείου του συνόλου εγγραφών.

FuzzyMappingsInfoStructure

Attributes

private OWLOntology ontology
private OWLDataFactory factory = OWLManager.createOWLOntologyManager().getOWLDataFactory()
private OWLClass concept[0..*]
private OWLObjectPropertyExpression role[0..*]
private owlThingHashSet
package Cache NQECache
package Cache RoleSetCache

Operations

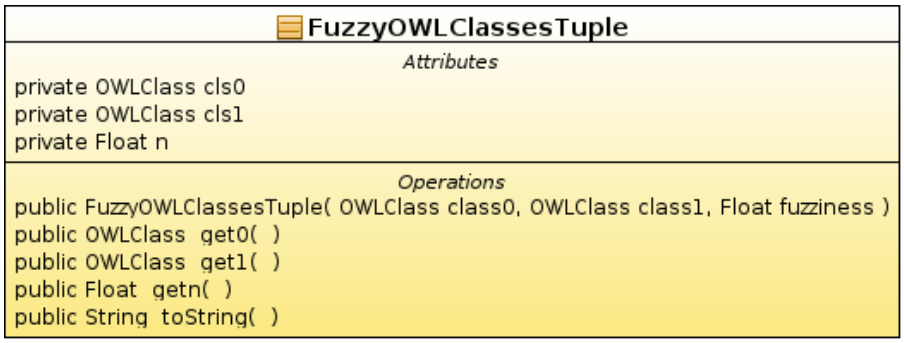
public FuzzyMappingsInfoStructure(OWLOntology ont)
private HashMap<Float, Set<FuzzyQueueElement>> neededQueueEntries(OWLDescription desc)
public boolean tupleExistsInRoleSet(FuzzyOWLClassesTuple t, OWLObjectPropertyExpression rl)
public boolean tupleExistsInRoleSet(OWLClass cls0, OWLClass cls1, OWLObjectPropertyExpression rl)
private int getIndexOf(OWLClass cls)
private int getIndexOf(OWLObjectPropertyExpression rl)
public OWLObjectPropertyExpression getRole(int i)
public OWLClass getConcept(int i)
public FuzzyQueueElement getFirstInQueue(int i)
public boolean isQueueEmpty(int i)
public boolean doesRoleSetContain(int i, FuzzyOWLClassesTuple t)
public boolean doesRoleSetContain(OWLObjectPropertyExpression rl, FuzzyOWLClassesTuple t)
public boolean doesConceptSetContain(int i, OWLClass cls)
public boolean doesConceptSetContain(int i, OWLClass cls, Float n)
public boolean doesConceptSetContainAll(int i, Set<OWLClass> set)
public Set<OWLClass> getConceptSet(OWLClass cls)
public Set<OWLClass> getConceptSetSafe(OWLClass cls)
public Set<FuzzyOWLClassesTuple> getRoleSet(OWLObjectPropertyExpression rl)
public Set<OWLClass> getRoleSetClassesSubsetByTuple1(OWLObjectPropertyExpression rl, OWLClass B)
public Set<OWLClass> getRoleSetClassesSubsetByTuple1(int i, OWLClass B)
public Map<OWLClass, Float> getRoleSetSubsetByTuple1(OWLObjectPropertyExpression rl, OWLClass B)
public Set<FuzzyOWLClassesTuple> getRoleSet(int i)
public void addTupleToRoleSet(OWLObjectPropertyExpression rl, FuzzyOWLClassesTuple t)
public void addToConceptSet(int i, OWLClass cls, Float n)
public void addNecessaryToQueue(int i, OWLDescription desc)
public void addNecessaryToQueue(OWLClass cls, OWLDescription desc)
public void addNecessaryToQueueWithMaxN(int i, OWLDescription desc, float maxN)
public void addNecessaryToQueueWithMaxN(OWLClass cls, OWLDescription desc, float maxN)
public int conceptSize( )
public Iterator conceptIterator( )
public Iterator roleIterator( )
public int getRoleSize( )
public int getQueuesSize( )
private void addClassRoleToReference(OWLClass B, OWLObjectPropertyExpression rl)
public HashSet<OWLObjectPropertyExpression> getRolesReferencingToClass(OWLClass B)
public Float getAxiomFuziness(OWLSubClassAxiom ax)
public Float getConceptSetEntryFuzziness(OWLClass setOfConcept, OWLClass entryOfConcept)
public Float getConceptSetEntryFuzziness(int i, OWLClass entryOfConcept)

Διάγραμμα 16: *Διάγραμμα UML της κλάσης Datatypes.FuzzyMappingsInfoStructure*

7.2.8 Κλάση FuzzyOWLClassesTuple

Η κλάση FuzzyOWLClassesTuple χρησιμοποιείται για την αναπαράσταση των 3-Tuples που απαιτούνται ως στοιχεία του συνόλου R κάποιου ονοματικού ρόλου μίας $f - \mathcal{EL}^+$ οντολο-

γίας. Όπως αναφέρεται και στη θεωρητική περιγραφή του αλγορίθμου, τα στοιχεία των συνόλων αυτών είναι ουσιαστικά τριάδες αποτελούμενες από δύο ονοματικές κλάσεις και έναν βαθμό βεβαιότητας, έχουν δε μορφή $\langle A, B, n \rangle$. Για την αναπαράστασή κάθε τριάδας υλοποιήθηκε η παρούσα κλάση, της οποίας το διάγραμμα UML είναι αυτό που απεικονίζεται παρακάτω:

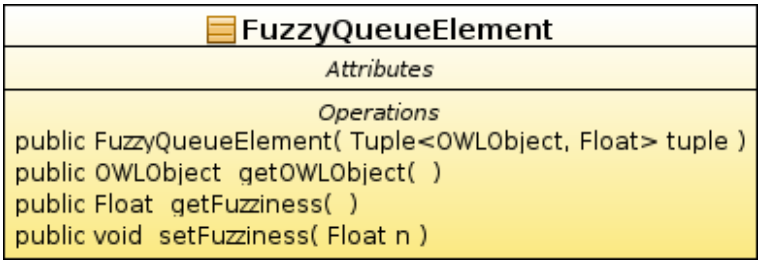


Διάγραμμα 17: *Διάγραμμα UML κλάσης Datatypes.FuzzyOWLClassesTuple*

Όπως γίνεται αντιληπτό, η λειτουργία της παρούσας κλάσης δε διαφέρει ιδιαίτερα από αυτή της CrispOWLClassesTuple: δίδει δυνατότητα δημιουργίας ενός αντικειμένου που θα περιέχει μία τριάδα αποτελούμενη από ένα ζεύγος κλάσεων και έναν βαθμό βεβαιότητας, καθώς και τη μετέπειτα ανάκτηση κάθε κλάσης του ζεύγους με τις get0() και get1() ή του βαθμού βεβαιότητας με την getn().

7.2.9 Κλάση FuzzyQueueElement

Η κλάση FuzzyQueueElement αποτελεί επέκταση της QueueElement και χρησιμοποιείται για την αποθήκευση των στοιχείων ουράς που απαιτούνται για τον υπολογισμό των αντιστοιχίσεων σε $f - \mathcal{EL}^+$ οντολογίες. Κάθε στοιχείο της ουράς περιέχει ένα ζεύγος OWLObject και βαθμού βεβαιότητας. Η υλοποίησή της είναι απλή, όπως διακρίνεται και από το UML διάγραμμά της:

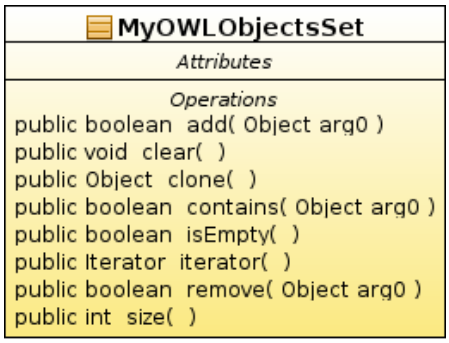


Διάγραμμα 18: *Διάγραμμα UML της κλάσης Datatypes.FuzzyQueueElement*

Η κλάση αυτή έχει μόνο τέσσερις μεθόδους: τη μέθοδο-κατασκευαστή που δημιουργεί ένα νέο στοιχείο ουράς, τη μέθοδο που ανακτά το OWLObject που περιλαμβάνει το στοιχείο αυτό, τη μέθοδο που ανακτά τον σχετικό βαθμό βεβαιότητας και τη μέθοδο που τον ενημερώνει – επανορίζει.

7.2.10 Κλάση MyOWLObjectsSet

Η κλάση αυτή επεκτείνει την κλάση java.util.HashSet. Δημιουργεί ένα HashSet στο οποίο μπορούν να εισαχθούν μόνο αντικείμενα ορισμένων συγκεκριμένων τύπων (βλέπε κώδικα μεθόδου add). Κατά τα άλλα δε διαφέρει από ένα οποιοδήποτε άλλο HashSet και οι υπόλοιπες μέθοδοι που υλοποιεί είναι αντίστοιχες. Χρησιμοποιείται μόνο για λόγους επιβεβαίωσης της ορθής λειτουργίας του κώδικα. Το διάγραμμα UML της κλάσης αυτής παρουσιάζεται παρακάτω:



Διάγραμμα 19: *Διάγραμμα UML κλάσης Datatypes.MyOWLObjectsSet*

Όπως διαπιστώνεται, η κλάση αυτή υλοποιεί τις βασικές μεθόδους οποιουδήποτε συνόλου.

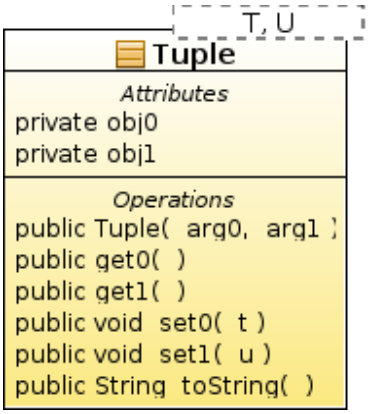
7.2.11 Κλάση QueueElement

Η αφηρημένη (abstract) αυτή κλάση αναπαριστά ένα γενικό στοιχείο ουράς, είτε για σαφή είτε για ασαφή οντολογία. Δεν υλοποιεί μεθόδους ή κάποια ιδιαίτερη λειτουργία πέραν αυτού.

7.2.12 Κλάση Tuple

Η παρούσα κλάση αποτελεί δομή δεδομένων με λειτουργία και συμπεριφορά παρόμοια με ενός 2-Tuple. Στον ορισμό της κλάσης δεν αναφέρονται οι τύποι των αντικειμένων που θα περιέχει το Tuple. Με τον τρόπο αυτό παρέχεται η ευελιξία στον προγραμματιστή να ορίσει μόνος του ζεύγη αντικειμένων καθορισμένων τύπων και να εκτελέσει απλές λειτουργίες πάνω στο Tuple.

Οι μέθοδοι που υλοποιεί η κλάση παρουσιάζονται στο ακόλουθο UML διάγραμμα:



Διάγραμμα 20: *Διάγραμμα UML για την κλάση Datatypes.Tuple*

Από το διάγραμμα αυτό διακρίνεται ότι οι τύποι των δύο στοιχείων δεν είναι προκαθορισμένοι, αλλά ο καθορισμός τους αφήνεται στον προγραμματιστή. Επίσης, υλοποιούνται απλές μέθοδοι κατασκευής νέου Tuple που περιέχει δύο αντικείμενα αλλά και ανάκτησης και ανάθεσης κάθε ενός από τα στοιχεία αυτά.

7.3 Πακέτο Utilities

Το πακέτο Utilities περιέχει τις παρακάτω κλάσεις:

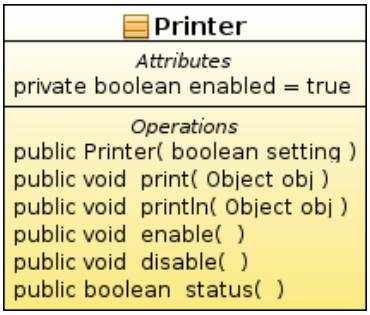
1. **Printer:** Η κλάση Printer δίνει τη δυνατότητα ενεργοποίησης/απενεργοποίησης των μηνυμάτων που τυπώνει κάθε τμήμα του προγράμματος.
2. **ScheduledStatPrint:** Η κλάση ScheduledStatPrint δίνει τη δυνατότητα περιοδικής εκτύπωσης του πλήθους των εγγραφών στις ουρές κατά το στάδιο του υπολογισμού των mappings.
3. **Timer:** Η κλάση αυτή δίνει τη δυνατότητα δημιουργίας ενός Timer, ο οποίος θα δέχεται τις εντολές start και stop και θα μπορεί να επιστρέψει το χρόνο που μεσολάβησε σε διάφορες μορφές.
4. **VirtualClassChecker:** Η κλάση αυτή δίνει τη δυνατότητα ελέγχου αν μία έννοια έχει ως όνομα έναν Μοναδικό Προσδιοριστή (Unique Identifier, UID), έχει δηλαδή παραχθεί και εισαχθεί αυτόματα στην οντολογία κατά το στάδιο της κανονικοποίησης.

Οι κλάσεις του παρόντος πακέτου παρουσιάζονται επιγραμματικά παρακάτω:

7.3.1 Κλάση Printer

Η κλάση Printer δίνει τη δυνατότητα ενεργοποίησης και απενεργοποίησης των μηνυμάτων που τυπώνει κάθε τμήμα του προγράμματος. Συγκεκριμένα, κάθε κλάση που χρειάζεται να εκτυπώνει μηνύματα δημιουργεί έναν δικό της Printer, τον οποίο και θα χρησιμοποιεί για

όλες τις εκτυπώσεις. Ο Printer επιτρέπει τις λειτουργίες print και println, ενώ έχει δύο καταστάσεις: την ενεργή και την ανενεργή. Οι καταστάσεις αυτές αντιστοιχούν στο εάν ο χρήστης επιθυμεί ή όχι τα σχετικά μηνύματα και πρέπει να ορίζονται από την καλούσα κλάση. Το UML διάγραμμα της παρούσας κλάσης απεικονίζεται στη συνέχεια:



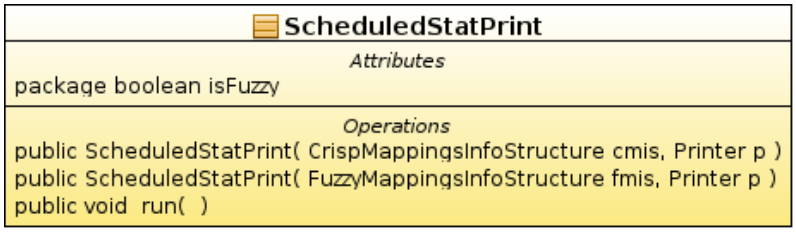
Διάγραμμα 21: *Διάγραμμα UML κλάσης Utilities.Printer*

Όπως γίνεται αντιληπτό από το διάγραμμα της κλάσης, παρέχονται οι μέθοδοι εκτύπωσης print και println, η δυνατότητα ενεργοποίησης ή απενεργοποίησης του εκτυπωτή (enable(), disable(), καθώς και η δυνατότητα ελέγχου της κατάστασής του (status()).

7.3.2 Κλάση ScheduledStatPrint

Η κλάση ScheduledStatPrint δίνει τη δυνατότητα περιοδικής εκτύπωσης του πλήθους των εγγραφών ουρών κατά το στάδιο του υπολογισμού των συνόλων αντιστοιχίσεων S και R. Χρησιμεύει στο να δίδεται μία εικόνα της πορείας της εκτέλεσης του προγράμματος προς το χρήστη, καθώς η διαδικασία υπολογισμού των αντιστοιχίσεων μπορεί να διαρκέσει αρκετό χρόνο.

Το διάγραμμα UML της παρούσας κλάσης είναι το παρακάτω:

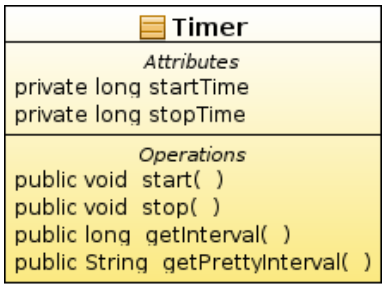


Διάγραμμα 22: *Διάγραμμα UML της κλάσης Utilities.ScheduledStatPrint*

Η παρούσα κλάση επεκτείνει την κλάση java.util.TimerTask. Συγκεκριμένα, η μέθοδος run() εκτελείται περιοδικά και τυπώνει το συνολικό μήκος των ουρών στο στάδιο των αντιστοιχίσεων.

7.3.3 Κλάση Timer

Η κλάση Timer δίνει τη δυνατότητα δημιουργίας ενός Timer (χρονομέτρου), ο οποίος θα δέχεται τις εντολές start και stop και θα μπορεί να επιστρέψει το χρόνο που μεσολάβησε σε διάφορες μορφές. Χρησιμοποιείται για την καταγραφή του χρόνου που απαιτείται για την ταξινόμηση μίας οντολογίας. Το UML διάγραμμα της κλάσης αυτής είναι το ακόλουθο:

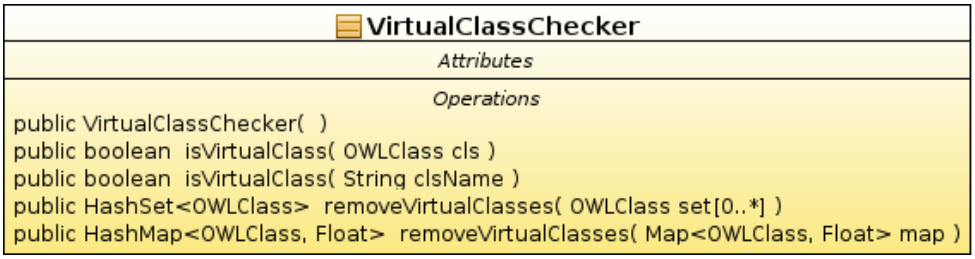


Διάγραμμα 23: *Διάγραμμα UML της κλάσης Utilities.Timer*

7.3.4 Κλάση VirtualClassChecker

Η κλάση αυτή δίνει τη δυνατότητα ελέγχου αν το όνομα μίας έννοιας έχει τη μορφή ενός UID (Unique Identifier), αν δηλαδή έχει παραχθεί και εισαχθεί αυτόματα στην οντολογία κατά το στάδιο της κανονικοποίησης. Χρησιμοποιείται ώστε να αγνοούνται οι αυτόματα εισαχθείσες στην οντολογία έννοιες κατά την εξαγωγή των πληροφοριών ταξινόμησης της οντολογίας.

Το UML διάγραμμα της παρούσας κλάσης απεικονίζεται παρακάτω:



Εικόνα 24: *Διάγραμμα UML της κλάσης Utilities.VirtualClassChecker*

Παρέχονται μέθοδοι ελέγχου αν μία κλάση έχει εισαχθεί αυτόματα (isVirtualClass), καθώς και μέθοδοι που αφαιρούν εγγραφές που αφορούν αυτόματα εισαχθείσες κλάσεις από σύνολα κλάσεων ή πληροφοριών ταξινόμησης (removeVirtualClasses).

8. Παραδείγματα Εκτέλεσης

Στην ενότητα αυτή θα επιδειχθεί ο τρόπος χρήση, λειτουργίας και εκτέλεσης της μηχανής συλλογιστικής, ενώ επίσης θα παρουσιαστούν και οι διάφορες μορφές εξόδου των υπολογισμών που αυτή υποστηρίζει. Συγκεκριμένα, θα αναφερθεί ο τρόπος ενεργοποίησης των διαφόρων λειτουργιών κάθε διεπαφής και θα περιγραφούν τα αποτελέσματα της εκτέλεσης κάθε μίας.

8.1 Μορφή Εκτελέσιμου Αρχείου

Το εκτελέσιμο αρχείο προσφέρεται στη μορφή του Java Archive (JAR). Για την επιτυχή εκτέλεσή του πρέπει στον φάκελο στον οποίο βρίσκεται να υπάρχει ένας υποφάκελος με όνομα lib, ο οποίος και να περιέχει τα Java Archives όλων των βιβλιοθηκών που χρησιμοποιούνται. Όλα αυτά περιλαμβάνονται στο συνοδευτικό CD, το οποίο βρίσκεται στο οπισθόφυλλο του παρόντος αντιτύπου.

Το πρόγραμμα έχει γραφεί χρησιμοποιώντας την 6η έκδοση της γλώσσας προγραμματισμού Java. Αυτό σημαίνει πως θα πρέπει και η εικονική μηχανή Java (Java Virtual Machine, JVM) που χρησιμοποιείται για την εκτέλεσή του να υποστηρίζει την έκδοση αυτή. Σε κάθε περίπτωση πάντως, ο κώδικας του προγράμματος είναι ελεύθερα διαθέσιμος και ο οποιοσδήποτε ενδιαφερόμενος μπορεί να κατασκευάσει δικό του εκτελέσιμο αρχείο ή να τον βελτιώσει / τροποποιήσει κατά το δοκούν.

8.2 Δυνατότητες Μηχανής Συλλογιστικής

Για να εκκινηθεί η μηχανή συλλογιστικής πρέπει να χρησιμοποιηθεί ο κατάλληλος εκκινητής Java (Java Launcher), ανάλογα με την εικονική μηχανή που χρησιμοποιείται. Για τις παρεχόμενες εικονικές μηχανές από το Sun JDK 6 και το OpenJDK 6, αυτό γίνεται με την εντολή:

```
java -jar fELpReasoner.jar
```

Στην περίπτωση που το πρόγραμμα κληθεί έτσι, εμφανίζει την παρακάτω έξοδο:

```
$ java -jar fELpReasoner.jar
fuzzy-EL+ Reasoner
Michalis Makaronides, 2009

Available Commands:
 -a FILE, --ask FILE                    Load queries from ontology FILE. Will
                                              output to stdout unless -oa is set.
 -d, --daemon                           Start as service (listening port 8080)
 -e A B, --equivalent A B               Check if class A is equivalent to class B
 -h, --help                             Display this help message
 -l FILE, --load=FILE                   Load and classify ontology FILE
 -nf, --no-fuzzy                        Instruct reasoner to ignore axiom
```

```
fuzziness
 -oao, --outputAnswersOntology          Output answers to the queries of the
                                              ontology specified by -a as an
                                              annotated ontology.
 -oc, --outputClassification            Output classification information for
                                              classes.
 -oio, --outputInferredOntology         Output inferred ontology in OWL/XML
                                              format.
 -ono, --outputNormalizedOntology       Output normalised ontology in OWL/XML
                                              format.
 -q, --quit                             No pause for manual query entry.
 -s A B, --subsumes A B                 Check if class A is  subsumed by class B
```

Το ίδιο κείμενο βοήθειας αυτό εμφανίζεται και αν κληθεί το πρόγραμμα με τα ορίσματα -h ή --help.

Γενικά, μπορεί να παρατηρηθεί πως κάθε όρισμα έχει δύο μορφές: μία συνεπτυγμένη, της οποίας το όνομα ξεκινά με μονή παύλα και και μία εκτεταμένη, της οποίας το όνομα ξεκινά με δύο παύλες. Και οι δύο μορφές αναπαράστασης είναι ισοδύναμες. Στη συνέχεια θα εξηγηθεί η λειτουργία κάθε ορίσματος με παραδείγματα. Η παρουσίαση που γίνεται για καθένα απ' αυτά αναφέρεται σε μία κλασική $\mathcal{EL}^+$ οντολογία, αλλά θα μπορούσε με τον ίδιο ακριβώς τρόπο να αναφέρεται και σε μία $f-\mathcal{EL}^+$. Στην περίπτωση αυτή βέβαια, οι αποκρίσεις της μηχανής συλλογιστικής θα περιελάμβαναν και βαθμούς βεβαιότητας μεταξύ των 0, 1 – πράγμα το οποίο στην περίπτωση μίας σαφούς οντολογίας δε συμβαίνει και εμφανίζονται μόνο οι δύο αυτές ακραίες τιμές.

1. -a FILE, --ask FILE

Το παραπάνω όρισμα χρησιμοποιείται προκειμένου να ορισθεί ένα αρχείο οντολογίας ερωτημάτων. Η οντολογία αυτή θα περιέχει ερωτήματα σε μορφή αξιωμάτων υπαγωγής ή ισοδυναμίας κλάσεων, των οποίων της απάντηση (βαθμός βεβαιότητας ισχύος) επιθυμούμε να λάβουμε από τη μηχανή συλλογιστικής. Τα ερωτήματα μπορεί να περιέχουν και σύνθετες κλάσεις (γενικευμένα αξιώματα υπαγωγής), όχι μόνο ονοματικές. Η μηχανή συλλογιστικής, αφού ταξινομήσει την αρχική οντολογία, ελέγχει ένα προς ένα τα ερωτήματα και εμφανίζει το βαθμό ισχύος τους στην οθόνη, εκτός και αν έχει χρησιμοποιηθεί και το όρισμα -oa.

Παράδειγμα Εκτέλεσης

Παρακάτω θα ζητήσουμε από τη μηχανή συλλογιστικής να ταξινομήσει μία οντολογία και να απεικονίσει στην οθόνη το βαθμό ισχύος των ερωτημάτων που περιέχονται σε μία άλλη οντολογία. Για τις ανάγκες του παραδείγματος ως νέα οντολογία έχει χρησιμοποιηθεί η οντολογία παραγόμενων αξιωμάτων, όπως προκύπτει από τη δυνατότητα 8.

Σημείωση: Οι χρόνοι που εμφανίζεται ότι απαιτούνται για την ταξινόμηση αυτών των οντολογιών δεν πρέπει να χρησιμοποιηθούν για σύγκριση με άλλα συστήματα συλλογιστικής. Αυτό συμβαίνει γιατί κατά την εκτέλεσή τους εκτελούνταν και πολλές άλλες εφαρμογές στον ίδιο υπολογιστή. Μετρήσεις αξιόπιστες για συγκρίσεις απόδοσης μεταξύ διαφορετικών μηχανών συλλογιστικής παρέχονται στο επόμενο κεφάλαιο, "Μετρήσεις και Συμπεράσματα" (σελ. 107).

Επίσης, τα δύο μηνύματα προειδοποίησης (Warnings) που εμφανίζονται αφορούν την Whirly Cache και οφείλονται σε παράλειψη από τους προγραμματιστές – κατασκευαστές της στον εσωτερικό της κώδικα, η οποία δεν επηρεάζει σε καμία περίπτωση τη λειτουργία της παρούσας μηχανής συλλογιστικής.

```
$ java -jar fELpReasoner.jar -l ~/NetBeansProjects/Ontologies/not-galen.owl -a ~/NetBeansProjects/Ontologies/not-galen-inferred.owl
Will load /home/michalis/NetBeansProjects/Ontologies/not-galen.owl
Will answer the queries of /home/michalis/NetBeansProjects/Ontologies/not-galen-inferred.owl
The ontology contains 4379 axioms, 2748 concept names and 413 role names.
This is an EL+ ontology.
Expressivity check: 28 msec
Normalisation: 2.081 sec
log4j:WARN No appenders could be found for logger (com.whirlycott.cache.CacheManager).
log4j:WARN Please initialize the log4j system properly.
Queues size: 1530
Mappings Computation: 5.36 sec
DAG Computation: 401 msec
Answering queries from file...
1.00000: SubClassOf(Infant Baby)
1.00000: SubClassOf(Membrane GenericInternalStructure)
1.00000: SubClassOf(Clinic SocialOrganisation)
... (συνεχίζει)
```

1. Παρατηρούμε πως; η μηχανή συλλογιστικής αναφέρει αναλυτικά ποιες ενέργειες εκτελεί κάθε στιγμή. Αφού ολοκληρώσει όλες τις απαιτούμενες για την ταξινόμηση της οντολογίας λειτουργίες αρχίζει να απαντά ένα προς ένα τα ερωτήματα της οντολογίας, αναφέροντας πρώτα τον βαθμό ισχύος και ύστερα το ερώτημα.

2. **-d, --daemon**

Αν διαπιστωθεί ύπαρξη αυτού του ορίσματος, τότε όλα τα υπόλοιπα ορίσματα αγνοούνται και το σύστημα ενεργοποιεί τη δυνατότητα απομακρυσμένης διεπαφής. Η δυνατότητα αυτή παρακολουθεί τη θύρα 8080 του τοπικού συστήματος και αναμένει αιτήσεις σύνδεσης. Για παράδειγμα, αν καλέσουμε τη μηχανή με την εντολή:

```
$ java  -jar fELpReasoner.jar -d
```

θα δεχθούμε την ακόλουθη έξοδο, στην οποία αναφέρεται το όνομα του τοπικού υπολογιστή (αν είναι διαθέσιμο) και η θύρα στην οποία αναμένει νέες συνδέσεις η υπηρεσία:

```
Server started
on M2-M1330:8080
```

Αν γίνει λοιπόν μία προσπάθεια σύνδεσης στη θύρα αυτή, θα λάβουμε το ακόλουθο μήνυμα:

```
$ telnet localhost 8080
Trying ::1...
Connected to localhost.
Escape character is '^]'.
* Connected to f-EL+ Reasoner!
* Supported commands:
*            load URL     Loads and classifies the specified ontology
*            sub A B      Ask whether class A is subsumed by class B
*            eq  A B      Ask whether class A is equivalent to class B
*            help         Display this help message
*            quit         Quit
```

Στο σημείο αυτό πρέπει να διευκρινισθεί για μία ακόμη φορά πως οι λειτουργίες που υποστηρίζονται από την απομακρυσμένη διεπαφή είναι ένα υποσύνολο των λειτουργιών όλου του συστήματος. Συγκεκριμένα, επειδή η διεπαφή αυτή προορίζεται κυρίως για χρήση από κάποιο πρόγραμμα και όχι από κάποιον άνθρωπο, υποστηρίζονται τρεις απλές λειτουργίες:

1. Πρώτον, η ανάκτηση και η ταξινόμηση απομακρυσμένης ή ακόμα και τοπικής οντολογίας.
2. Δεύτερον, ο έλεγχος υπαγωγής ονοματικών κλάσεων και
3. τρίτον, ο έλεγχος ισοδυναμίας ονοματικών κλάσεων.

Παρακάτω θα παρουσιαστεί μία τυπική σειρά απομακρυσμένων αιτημάτων και απαντήσεων της μηχανής συλλογιστικής, όπου με **έντονη** γραμματοσειρά απεικονίζονται τα μηνύματα που στέλνονται από τον χρήστη:

```
load http://lat.inf.tu-dresden.de/~meng/ontologies/not-galen.owl
>*-1/5 Loaded Ontology
*-2/5 Is of supported expressivity
*-3/5 Preprocessed
*-4/5 Normalized
*-5/5 Subsumer sets computed
OK-Ontology classified!
sub Neisseria GramNegativeBacterium
>1.00000
sub GramNegativeBacterium Neisseria
>0.00000
eq Cobalamin VitaminB12
>1.00000
eq NotExistant1 NotExistant2

>ERROR-Class NotExistant1 not found!
quit
>BYE
Connection closed by foreign host.
```

Στο παραπάνω παράδειγμα ο χρήστης ζητά αρχικά την ανάκτηση και ταξινόμηση μίας οντολογίας (εντολή load). Στη συνέχεια, ζητούνται ορισμένες πληροφορίες βαθμού βεβαιότητας ισχύος υπαγωγής και ισοδυναμίας κλάσεων (εντολές sub, eq), όπου η μηχανή αποκρίνεται με τους αντίστοιχους βαθμούς ισχύος ή με σφάλμα σε περίπτωση που δεν υπάρχει κλάση με τέτοιο όνομα. Τέλος, ο χρήστης αποστέλλει εντολή τερματισμού της συνεδρίας (εντολή quit) και η σύνδεση τερματίζεται.

3. -e A B, --equivalent A B

Με το όρισμα αυτό ορίζεται ένα ζεύγος κλάσεων, οι οποίες θα ελεγχθούν για ισοδυναμία. Το αποτέλεσμα του ελέγχου θα εμφανιστεί στην οθόνη μετά το πέρας της διαδικασίας ταξινόμησης της οντολογίας. Υποστηρίζεται η δυνατότητα ορισμού πολλαπλών ζευγών εννοιών, με πολλαπλά ορίσματα του τύπου αυτού. Παράδειγμα εκτέλεσης θα δοθεί μαζί με τη δυνατότητα 11.

4. -h, --help

Εμφάνιση κειμένου βοήθειας για τις διάφορες λειτουργίες της μηχανής συλλογιστικής.

5. -nf, --no-fuzzy

Αν χρησιμοποιηθεί αυτό το όρισμα, τότε η μηχανή συλλογιστικής καλείται να αγνοήσει την πιθανή ύπαρξη επισημειώσεων βαθμών βεβαιότητας στην οντολογία. Έτσι, κάθε οντολογία θεωρείται ως σαφής (crisp). Η λειτουργία αυτή είναι χρήσιμη κυρίως για να είναι άμεσα συγκρίσιμες οι μετρήσεις χρόνου εκτέλεσης μεταξύ της παρούσας μηχανής συλλογιστικής και άλλων, όπως του CEL, οι οποίες δεν υποστηρίζουν ασαφείς περιγραφικές λογικές. Αν δε χρησιμοποιηθεί το όρισμα αυτό, τότε σε οντολογίες οι οποίες περιέχουν πολλές επισημειώσεις η μηχανή συλλογιστικής θα χρειαστεί παραπάνω χρόνο στο αρχικό στάδιο του ελέγχου της εκφραστικότητας, καθώς θα είναι αναγκασμένη να εξετάσει την πιθανή χρησιμότητα κάθε επισημείωσης ως ασαφούς βαθμού βεβαιότητας για κάποιο αξίωμα. Αυτό θα έχει ως αποτέλεσμα να φαίνεται η λειτουργία της μηχανής συλλογιστικής βραδύτερη σε σχέση με άλλες μηχανές, οι οποίες υλοποιούν όμως ένα υποσύνολο των λειτουργιών.

6. -oao, --outputAnswersOntology

Το όρισμα αυτό μπορεί να χρησιμοποιηθεί μόνο σε συνδυασμό με το όρισμα 1 (-a). Η χρήση του έχει ως αποτέλεσμα την εξαγωγή της οντολογίας ερωτημάτων με επισημειωμένο τον βαθμό βεβαιότητας ισχύος κάθε ερωτήματος – αξιώματος. Αν ο βαθμός ισχύος ισούται με τη μονάδα (απόλυτη βεβαιότητα), τότε αυτός δεν καταγράφεται στην οντολογία αλλά καταγράφεται μόνο το σχετικό αξίωμα. Η οντολογία αυτή θα εξαχθεί είτε στον ίδιο φάκελο που βρίσκεται η οντολογία προς ταξινόμηση, αν αυτός ανήκει στο τοπικό σύστημα αρχείων, είτε στον φάκελο από τον οποίο έχει κληθεί το πρόγραμμα, σε περί-

πτωση που η οντολογία ερωτημάτων ανακτάται μέσω HTTP. Σε κάθε περίπτωση το αρχείο αυτό θα έχει ως όνομα τη συνένωση του ονόματος της οντολογίας προς ταξινόμηση (χωρίς την κατάληψη ".owl") με την συμβολοσειρά "-qnas.owl" (Questions and Answers, Ερωτήσεις και Απαντήσεις). Για παράδειγμα, αν η οντολογία βρίσκεται στο αρχείο "test.owl", τότε η οντολογία απαντήσεων θα βρίσκεται στο "test-qnas.owl".

Παράδειγμα Εκτέλεσης

Μπορούμε να ζητήσουμε την εξαγωγή της οντολογίας που περιέχει τις απαντήσεις σε κάθε ερώτημα της οντολογίας ερωτημάτων ως επισημειώσεις για κάθε ερώτημα – αξίωμα με την ακόλουθη εντολή:

```
$ java -jar fELpReasoner.jar -l ~/NetBeansProjects/Ontologies/not-galen.owl -a ~/NetBeansProjects/Ontologies/not-galen-inferred.owl -oao -q
```

Στο τέλος της διαδικασίας αυτής το αρχείο "not-galen-qnas.owl" θα περιέχει την οντολογία με τις απαντήσεις – επισημειώσεις.

7. **-oc, --outputClassification**

Αν χρησιμοποιηθεί το όρισμα αυτό, τότε εξάγονται τα δεδομένα ταξινόμησης για κάθε κλάση σε ένα αρχείο κειμένου. Τα δεδομένα ταξινόμησης για κάθε κλάση περιέχουν το όνομα της κλάσης, τις άμεσες κλάσεις-γονείς της, τις ισοδύναμες κλάσεις της και τις άμεσες κλάσεις-παιδιά της. Το αρχείο εξόδου θα έχει ως όνομα τη συνένωση του ονόματος της οντολογίας προς ταξινόμηση (χωρίς την κατάληψη ".owl") με την συμβολοσειρά ".txt". Για παράδειγμα, αν η οντολογία βρίσκεται στο αρχείο "test.owl", τότε οι πληροφορίες ταξινόμησης θα περιέχονται στο "test.txt".

Για την επιτυχή ολοκλήρωση της διαδικασίας αυτής είναι απαραίτητος ο υπολογισμός του συνεκτικού ακυκλικού γράφου υπαγωγής (subsumption DAG). Έτσι, στην περίπτωση που διαπιστωθεί η ύπαρξη του ορίσματος -oc η μηχανή συλλογιστικής αυτόματα υπολογίζει και τον γράφο υπαγωγής.

Παράδειγμα Εκτέλεσης

Μπορούμε να ζητήσουμε την εξαγωγή των πληροφοριών ταξινόμησης για κάθε κλάση με την ακόλουθη εντολή:

```
$ java -jar fELpReasoner.jar -l ~/NetBeansProjects/Ontologies/not-galen.owl -oc -q
```

Στο τέλος αυτής της διαδικασίας το αρχείο "not-galen.txt" θα περιέχει τις πληροφορίες ταξινόμησης, σε μορφή ίδια με αυτή που παρουσιάζεται στο παράδειγμα 9.

8. **-oio, --outputInferredOntology**

Με χρήση του ορίσματος αυτού δηλώνεται η επιθυμία εξαγωγής της αρχικής οντολογίας, συμπληρωμένης με όλα τα απαιτούμενα αξιώματα υπαγωγής και ισοδυναμίας κλάσεων που απουσιάζουν έτσι ώστε η ταξινόμησή της να είναι πλήρως καταγεγραμμένη. Η επιλογή αυτή είναι χρήσιμη για δύο λόγους. Κατά πρώτον, για να μπορεί στη συνέχεια να επεξεργαστεί την οντολογία ένας χρήστης σε κάποιο γραφικό εργαλείο, όπως το Protégé ή το σύστημα FReS και κατά δεύτερον, για να είναι δυνατός ο έλεγχος της ορθής λειτουργίας της μηχανής συλλογιστικής. Συγκεκριμένα, αν όλα τα αξιώματα ταξινόμησης έχουν συμπληρωθεί στην οντολογία τότε οποιαδήποτε μετέπειτα προσπάθεια ταξινόμησής της με χρήση κάποιας άλλης μηχανής συλλογιστικής δε θα πρέπει να είναι σε θέση να ανακαλύψει νέες σχέσεις ταξινόμησης. Αν είναι σε θέση, τότε σημαίνει πως η αρχική μηχανή συλλογιστικής δεν ταξινομεί ορθά την οντολογία..

Το αρχείο εξόδου θα έχει ως όνομα τη συνένωση του ονόματος της οντολογίας προς ταξινόμηση (χωρίς την κατάληψη ".owl") με την συμβολοσειρά "-inferred.owl". Για παράδειγμα, αν η οντολογία βρίσκεται στο αρχείο "test.owl", τότε η παραγόμενη οντολογία θα βρίσκεται στο "test-inferred.owl".

Για την επιτυχή ολοκλήρωση της διαδικασίας αυτής είναι απαραίτητος ο υπολογισμός του συνεκτικού ακυκλικού γράφου υπαγωγής (subsumption DAG). Έτσι, στην περίπτωση που διαπιστωθεί η ύπαρξη του ορίσματος -oio η μηχανή συλλογιστικής αυτόματα υπολογίζει και τον γράφο υπαγωγής.

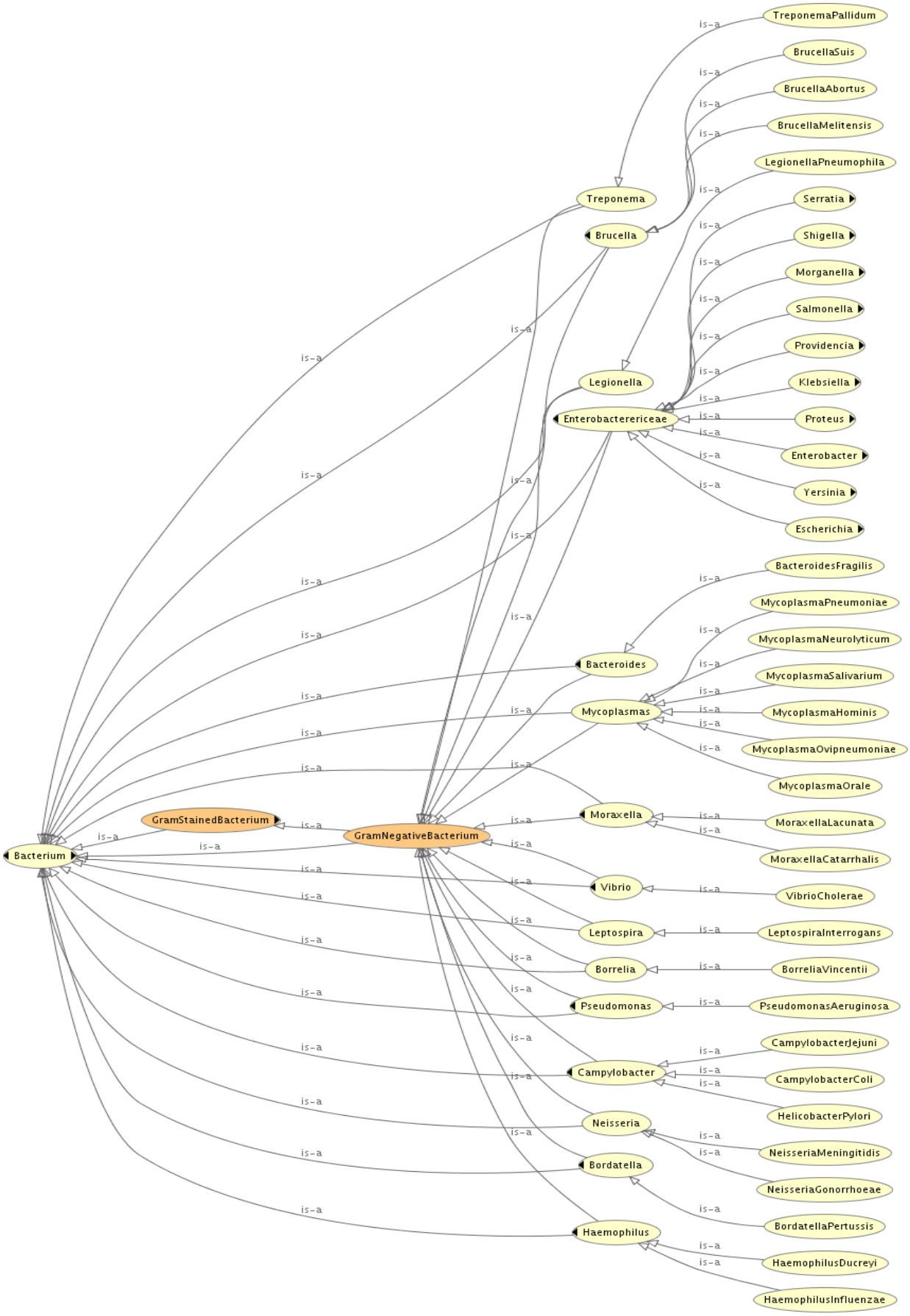
<u>Παράδειγμα Εκτέλεσης</u>

Μπορούμε να ζητήσουμε την εξαγωγή της παραγόμενης οντολογίας με την παρακάτω εντολή:

```
$ java -jar fELpReasoner.jar -l ~/NetBeansProjects/Ontologies/not-galen.owl -oio -q
```

Μετά το τέλος του προγράμματος το αρχείο "not-galen-inf.owl" θα περιέχει την παραγόμενη οντολογία. Ως ενδεικτικό της λειτουργίας του προγράμματος και της χρησιμότητας της δυνατότητας αυτής, θα παρουσιαστεί το παραγόμενο υποδέντρο για την κλάση "GramNegativeBacterium" της οντολογίας "Not-Galen" (βλ. "Ιατρική Βάση Γνώσης GALEN", σελ. 108), τόσο για την αρχική οντολογία όσο και για την παραγόμενη, η οποία περιέχει καταγεγραμμένες όλες τις σχέσεις μεταξύ των αναφερόμενων κλάσεων. Για την αναπαράσταση αυτή χρησιμοποιήθηκε το εργαλείο "OWLViz" του Protégé 4.0.

Διάγραμμα 25: *Αναπαράσταση Έννοιας GramNegativeBacterium βάσει Αρχικής Οντολογίας*


Bacterium
GramStainedBacterium
GramNegativeBacterium
Treponema
Brucella
Legionella
Enterobacterericeae
Bacteroides
Mycoplasmas
Moraxella
Vibrio
Leptospira
Borrelia
Pseudomonas
Campylobacter
Neisseria
Bordatella
Haemophilus
TreponemaPallidum
BrucellaSuis
BrucellaAbortus
BrucellaMelitensis
LegionellaPneumophila
Serratia
Shigella
Morganella
Salmonella
Providencia
Klebsiella
Proteus
Enterobacter
Yersinia
Escherichia
BacteroidesFragilis
MycoplasmaPneumoniae
MycoplasmaNeurolyticum
MycoplasmaSalivarium
MycoplasmaHominis
MycoplasmaOvipneumoniae
MycoplasmaOrale
MoraxellaLacunata
MoraxellaCatarrhalis
VibrioCholerae
LeptospiraInterrogans
BorreliaVincentii
PseudomonasAeruginosa
CampylobacterJejuni
CampylobacterColi
HelicobacterPylori
NeisseriaMeningitidis
NeisseriaGonorrhoeae
BordatellaPertussis
HaemophilusDucreyi
HaemophilusInfluenzae
is-a

Στην προηγούμενη σελίδα απεικονίζεται η ίδια αναπαράσταση, αλλά βάσει της παραγόμενης οντολογίας. Παρατηρούμε ότι πολυάριθμες έννοιες καταχωρούνται ως υπαγόμενες έννοιες της GramNegativeBacterium, ενώ ταυτόχρονα και για λόγους πληρότητας διατηρούνται και οι αρχικές συσχετίσεις που υπήρχαν στην οντολογία.

9. **-ono, --outputNormalizedOntology**

Αν χρησιμοποιηθεί αυτό το όρισμα, τότε δηλώνεται η επιθυμία εξαγωγής της κανονικοποιημένης οντολογίας, με όλες τις αυτόματα εισαχθείσες κλάσεις, σε κάποιο αρχείο. Η λειτουργία αυτή είναι χρήσιμη μόνο για έλεγχο της ορθής λειτουργίας της μηχανής συλλογιστικής.

Το αρχείο εξόδου θα έχει ως όνομα τη συνένωση του ονόματος της οντολογίας προς ταξινόμηση (χωρίς την κατάληψη ".owl") με την συμβολοσειρά "-norm.owl". Για παράδειγμα, αν η οντολογία βρίσκεται στο αρχείο "test.owl", τότε η παραγόμενη οντολογία θα βρίσκεται στο "test-norm.owl".

<u>Παράδειγμα Εκτέλεσης</u>

Μπορούμε να ζητήσουμε την εξαγωγή της κανονικοποιημένης οντολογίας με την ακόλουθη εντολή:

```
$ java -jar fELpReasoner.jar -l ~/NetBeansProjects/Ontologies/not-galen.owl -ono -q
```

Μετά το τέλος της εκτέλεσης του προγράμματος, το αρχείο "not-galen-norm.owl" θα περιέχει την κανονικοποιημένη οντολογία.

10. **-q, --quit**

Με το όρισμα αυτό δηλώνεται πως μετά την εκτέλεση όλων των ενεργειών και την απάντηση όλων των ερωτημάτων που έχει ορίσει ο χρήστης το πρόγραμμα θα σταματήσει τη λειτουργία του. Σε αντίθετη περίπτωση και μετά το τέλος όλων των λειτουργιών δίδεται η δυνατότητα στο χρήστη να εισάγει από το πληκτρολόγιο τμήμα ονόματος κάποιας έννοιας και να λάβει πληροφορίες για όσες έννοιες περιέχουν το τμήμα αυτό στο όνομά τους. Οι πληροφορίες που παρέχονται περιλαμβάνουν το πλήρες όνομα της κλάσης, τις άμεσες κλάσεις-γονείς της, τις ισοδύναμες κλάσεις της και τις άμεσες κλάσεις-παιδιά της.

Για την επιτυχή ολοκλήρωση της διαδικασίας αυτής και την παροχή των πληροφοριών ταξινόμησης είναι απαραίτητος ο υπολογισμός του συνεκτικού ακυκλικού γράφου υπαγωγής (subsumption DAG). Έτσι, στην περίπτωση που δε διαπιστωθεί η ύπαρξη του ορίσματος -q η μηχανή συλλογιστικής αυτόματα υπολογίζει και τον γράφο υπαγωγής.

Παράδειγμα Εκτέλεσης

Στο παράδειγμα αυτό θα παρουσιαστεί η διαδικασία υποβολής ερωτημάτων για κλάσεις από το πληκτρολόγιο, μετά το τέλος της πορείας συλλογιστικής. Αυτή είναι και η τυπική ενέργεια της μηχανής μετά την εκτέλεση όλων των λειτουργιών, αν δεν έχει χρησιμοποιηθεί το παρόν όρισμα.

```
$ java -jar fELpReasoner.jar -l ~/NetBeansProjects/Ontologies/not-galen.owl
Will load /home/michalis/NetBeansProjects/Ontologies/not-galen.owl
The ontology contains 4379 axioms, 2748 concept names and 413 role names.
This is an EL+ ontology.
Expressivity check: 76 msec
Normalisation: 939 msec
log4j:WARN No appenders could be found for logger
(com.whirlycott.cache.CacheManager).
log4j:WARN Please initialize the log4j system properly.
Queues size: 1403
Mappings Computation: 4.067 sec
DAG Computation: 303 msec


Will now accept queries. Quit with 'q'.

Query info for class:
neisseria
Class name:    Neisseria
Parents:       [GramNegativeBacterium]
Equivalents:   []
Children:      [NeisseriaMeningitidis, NeisseriaGonorrhoeae]

Class name:    NeisseriaGonorrhoeae
Parents:       [Neisseria]
Equivalents:   []
Children:      []

Class name:    NeisseriaMeningitidis
Parents:       [Neisseria]
Equivalents:   []
Children:      []



Query info for class:
q
```

Στο παραπάνω παράδειγμα φαίνεται η απόκριση της μηχανής για μία είσοδο του χρήστη. Οι χαρακτήρες που πληκτρολογούνται από τον χρήστη απεικονίζονται εδώ με **έντονη** γραμματοσειρά.

11. **-s A B, --subsumes A B**

Με το όρισμα αυτό ορίζεται ένα ζεύγος κλάσεων, οι οποίες και θα ελεγχθούν για υπαγωγή της πρώτης στη δεύτερη μετά το τέλος της διαδικασίας ταξινόμησης. Το αποτέλεσμα του ελέγχου θα εμφανιστεί στην οθόνη. Υποστηρίζεται η δυνατότητα ορισμού πολλαπλών ζευγών εννοιών, με πολλαπλά ορίσματα του τύπου αυτού.

Παράδειγμα Εκτέλεσης

Στο παράδειγμα αυτό θα ζητήσουμε από τη μηχανή συλλογιστικής να ελέγξει την ισοδυναμία και την υπαγωγή δύο ζευγών υπαρκτών κλάσεων και ενός ζεύγους ανυπάρκτων κλάσεων.

```
$ java -jar fELpReasoner.jar -l ~/NetBeansProjects/Ontologies/not-
galen.owl -e VitaminB12 Cobalamin -s Neisseria
GramNegativeBacterium -s NotExistant1 NotExistant2
Will load /home/michalis/NetBeansProjects/Ontologies/not-galen.owl
The ontology contains 4379 axioms, 2748 concept names and 413 role names.
This is an EL+ ontology.
Expressivity check: 77 msec
Normalisation: 957 msec
log4j:WARN No appenders could be found for logger
(com.whirlycott.cache.CacheManager).
log4j:WARN Please initialize the log4j system properly.
Queues size: 1345
Mappings Computation: 4.143 sec
DAG Computation: 347 msec
1.00000: Neisseria <= GramNegativeBacterium
Class NotExistant1 not found!
1.00000: VitaminB12 == Cobalamin
```

Όπως γίνεται αντιληπτό, η μηχανή απαντά όλα τα ερωτήματα και εμφανίζει ειδικό μήνυμα αν μία κλάση δεν υπάρχει στην οντολογία.

8.3 Εκτέλεση για Ασαφή Οντολογία

Η λειτουργία και η μορφή εξαγωγής των αποτελεσμάτων της μηχανής συλλογιστικής δε διαφέρει ανάλογα με το αν η οντολογία που εισήχθη προς ταξινόμηση είναι σαφής ή ασαφής. Η βασικότερη διαφορά είναι το ότι οι βαθμοί βεβαιότητας που χρησιμοποιούνται για να δηλώσουν κατά πόσον ισχύει ένα αξίωμα ή ερώτημα ανήκουν στο $\{0, 1\}$ για σαφείς οντολογίες και στο $[0, 1]$ για ασαφείς, αλλά τόσο τα σημεία στα οποία εμφανίζονται βαθμοί βεβαιότητας όσο και η ίδια η μορφή των βαθμών αυτών παραμένει αμετάβλητη.

Επίσης, σημαντικό είναι να τονισθεί πως δεν υπάρχουν διαθέσιμες εκτενείς ασαφείς οντολογίες σε Περιγραφική Λογική $f - \mathcal{EL}^+$. Για τον λόγο αυτό και προκειμένου να είναι δυνατός ο έλεγχος του τμήματος ασαφούς συλλογιστικής της μηχανής, προστέθηκαν ασαφείς επισημειώσεις σε αξιώματα κλασικών περιγραφικών λογικών $\mathcal{EL}^+$, μετατρέποντάς τις σε $f - \mathcal{EL}^+$.

Οι τροποποιηθείσες αυτές οντολογίες δεν έχουν βέβαια κάποιο φυσικό νόημα, είναι όμως πολύτιμες για έλεγχο της πορείας συλλογισμών. Η έξοδος μίας ενδεικτικής εκτέλεσης παρουσιάζεται στη συνέχεια:

```
$ java -jar fELpReasoner.jar -l ~/NetBeansProjects/Ontologies/not-galen-fuzzy.owl
Will load /home/michalis/NetBeansProjects/Ontologies/not-galen-fuzzy.owl
The ontology contains 4382 axioms, 2748 concept names and 413 role names.
This is a Fuzzy EL+ ontology.
Expressivity check: 55 msec
Normalisation: 1.084 sec
log4j:WARN No appenders could be found for logger
(com.whirlycott.cache.CacheManager).
log4j:WARN Please initialize the log4j system properly.
Queues size: 1336
Mappings Computation: 3.519 sec
DAG Computation: 391 msec


Will now accept queries. Quit with 'q'.

Query info for class:
TopCategory
Class name:     TopCategory
Parents:        Thing (1.0),
Equivalents:
Children:       DomainCategory (0.9),



Query info for class:
q
```

Παρατηρούμε πως η μηχανή συλλογιστικής αναφέρει τον κατάλληλο βαθμό βεβαιότητας ισχύος. Αυτή η συμπεριφορά μπορεί να παρατηρηθεί σε όλες τις περιπτώσεις αποκρίσεων της οντολογίας, καθώς οι διαφορετικοί τρόποι απαντήσεων είναι συγκεκριμένοι και ανεξάρτητοι της ασάφειας ή μη της ταξινομούμενης οντολογίας.

9. Μετρήσεις και Συμπεράσματα

Στο παρόν κεφάλαιο θα παρουσιαστεί αρχικά η πορεία που ακολουθήθηκε προκειμένου να διαπιστωθεί η ορθή υλοποίηση και λειτουργία του αλγορίθμου συλλογιστικής. Στη συνέχεια, θα παρουσιαστούν μετρήσεις του χρόνου εκτέλεσης που απαιτείται από την υλοποιηθείσα μηχανή συλλογιστικής και ορισμένες άλλες, ευρέως διαδεδομένες, για την ταξινόμηση ορισμένων εκτεταμένων οντολογιών οι οποίες χρησιμοποιούνται συχνά στη βιβλιογραφία για τον έλεγχο της απόδοσης μηχανών συλλογιστικής ([13], [14]). Η παρουσίαση των μετρήσεων θα συνοδευτεί από ανάλυση αυτών και των αιτίων που προκαλούν τις διάφορες μεταβολές. Τέλος, θα αναφερθούν τα συμπεράσματα που αποκομίσθηκαν από τη γενική προσπάθεια υλοποίησης και τα αποτελέσματα εκτέλεσης της μηχανής συλλογιστικής, καθώς και προτάσεις για τις κατευθύνσεις στις οποίες κρίνεται σκόπιμο να κινηθούν μελλοντικές βελτιώσεις και επεκτάσεις.

9.1 Πορεία Ελέγχου Ορθής Λειτουργίας

Για τον έλεγχο της μηχανής συλλογιστικής χρησιμοποιήθηκαν τέσσερις βασικές οντολογίες εισόδου. Οι οντολογίες αυτές είναι επαρκώς πολύπλοκες και περιέχουν σημαντικό αριθμό αξιωμάτων, εννοιών και ρόλων, γεγονός που τις καθιστά κατάλληλες και για χρήση ως οντολογίες ελέγχου της ταχύτητας ταξινόμησης μεταξύ διαφορετικών μηχανών συλλογιστικής. Χρησιμοποιήθηκαν δύο μεθοδολογίες ελέγχου ορθότητας:

1. Εισαγωγή της οντολογίας στο Protégé 4.0, ταξινόμησή της με χρήση των μηχανών συλλογιστικής FaCT++ και Pellet και στη συνέχεια αποθήκευση των παραγόμενων αξιωμάτων σε μία νέα οντολογία. Ύστερα, γίνεται προσπάθεια ταξινόμησης της αρχικής οντολογίας και από την υλοποιηθείσα μηχανή συλλογιστικής, με χρήση της νέας οντολογίας που προήλθε από τις άλλες μηχανές συλλογιστικής ως οντολογία ερωτημάτων προς απάντηση. Η λειτουργία της μηχανής συλλογιστικής μπορεί να θεωρηθεί ορθή εφ' όσον αυτή είναι σε θέση να απαντήσει θετικά σε όλα τα ερωτήματα. Η αξιοπιστία της θεωρείται τόση, όση αυτή της πληρότητας και της ορθότητας των εξαγόμενων πληροφοριών ταξινόμησης από τις μηχανές FaCT++ και Pellet. Όμως, επειδή και οι οντολογίες που χρησιμοποιήθηκαν για τον έλεγχο είναι πολύ διαδεδομένες στη βιβλιογραφία, αλλά και οι μηχανές συλλογιστικής έχουν ελεγχθεί αρκετά από τις ερευνητικές ομάδες που δραστηριοποιούνται στον χώρο των τεχνολογιών γνώσης, μπορεί αυτή η μεθοδολογία ελέγχου να θεωρηθεί αξιόπιστη με πολύ ικανοποιητικό βαθμό.
2. Δειγματοληπτικός έλεγχος των παραγόμενων πληροφοριών ταξινόμησης με τα αντίστοιχα αποτελέσματα του CEL (version 1.0 build 8) και άλλων μηχανών συλλογιστικής.

Η επιτυχής ολοκλήρωση των δύο αυτών μεθοδολογιών ελέγχου, σε συνδυασμό με τη δημιουργία και ταξινόμηση μικρών οντολογιών ελέγχου, οι οποίες περιέχουν αξιώματα συγκεκριμένου τύπου τα οποία δεν εμφανίζονται σε ικανοποιητικό βαθμό στις προαναφερθείσες οντολογίες – όπως οι αλυσίδες ρόλων – αποδεικνύουν στην πράξη την ορθή λειτουργία του τμήματος σαφούς συλλογιστικής της μηχανής αλλά και ενός μεγάλου τμήματος του ασαφούς αλγορίθμου.

Όσον αφορά το τμήμα ασαφούς συλλογιστικής, πρέπει να σημειωθεί πως η υλοποίησή του ξεκίνησε αφού ολοκληρώθηκε η υλοποίηση και ο έλεγχος του σαφούς τμήματος. Δεδομένου ότι οι δύο αλγόριθμοι παρουσιάζουν σημαντική ομοιότητα, ενώ επίσης το ασαφές σύστημα υλο-

ποιήθηκε ουσιαστικά ως επέκταση και συμπλήρωση του ελεγμένου κώδικα του σαφούς τμήματος, χρησιμοποιήθηκε η ακόλουθη μεθοδολογία ελέγχου ορθότητας:

Τροποποιήθηκαν οι τέσσερις βασικές και οι λοιπές μικρότερες αλλά αυξημένης πολυπλοκότητας οντολογίες, έτσι ώστε να προστεθούν ασαφείς βαθμοί βεβαιότητας ισχύος σε ορισμένα γενικευμένα ή μη αξιώματα υπαγωγής. Οι βαθμοί αυτοί μπορεί να μην είχαν κάποιο φυσικό νόημα, ήταν όμως κατάλληλοι για την επιβεβαίωση της ορθής λειτουργίας του συστήματος. Στη συνέχεια ελέγχθηκε αν η ύπαρξη των βαθμών αυτών οδηγεί σε λογικά συμπεράσματα ως προς τους βαθμούς βεβαιότητας, όπως αυτά που θα προέκυπταν διαισθητικά ή κατά την εκτέλεση του αλγορίθμου "με το χέρι". Και αυτή η μεθοδολογία ελέγχου ολοκληρώθηκε επιτυχώς.

9.2 Περιγραφή Οντολογιών Εισόδου

Στην ενότητα αυτή θα δοθούν περισσότερα στοιχεία για τις τέσσερις βασικές οντολογίες εισόδου. Οι τέσσερις οντολογίες αυτές χρησιμοποιούνται και για συγκρίσεις ταχύτητας ταξινόμησης μεταξύ των διαφόρων μηχανών συλλογιστικής, ενώ είναι οι ίδιες που χρησιμοποιήθηκαν και για την σύγκριση απόδοσης του CEL με άλλες μηχανές συλλογιστικής στα [13], [14]. Οι οντολογίες ελήφθησαν από το [27] σε μορφή OWL. Όλες οι χρησιμοποιηθείσες οντολογίες αναφέρονται σε βιοϊατρικές εφαρμογές, καθώς όπως έχει αναφερθεί οι εφαρμογές αυτές αποτελούν ένα πολύ σημαντικό τμήμα του πεδίου εφαρμογής της οικογένειας περιγραφικών λογικών $\mathcal{EL}$. Μία πολύ σημαντική βιοϊατρική οντολογία, η SNOMED-CT (Systematized Nomenclature of Medicine – Clinical Terms) ([16]), δεν ήταν δυνατό να χρησιμοποιηθεί καθώς δε διανέμεται ελεύθερα.

9.2.1 Ιατρική Βάση Γνώσης GALEN

Η οντολογία αυτή αναπτύχθηκε με στόχο το διαμοιρασμό και την κοινή χρήση ιατρικών πληροφοριών. Αρχικά αναπτύχθηκε στη γλώσσα GRAIL και εν συνεχεία καταγράφηκε και σε Περιγραφικές Λογικές ([4]). Στην καταγραφή αυτή σε Περιγραφικές Λογικές, η GALEN αρκείται στην εκφραστικότητα της $\mathcal{EL}$, προσαυξημένη με Γενικευμένα Αξιώματα Υπαγωγής (GCIs), ιεραρχίες ρόλων, μεταβατικούς ρόλους, functional ρόλους και αντίστροφους ρόλους, δηλαδή στην εκφραστικότητα της $\mathcal{ELHI}f_{R^+}$. Δεδομένου ότι η Περιγραφική Λογική $\mathcal{EL}+$ που υποστηρίζει η παρούσα μηχανή συλλογιστικής δεν υποστηρίζει αντίστροφους και functional ρόλους, η χρησιμοποιούμενη οντολογία έχει υποστεί ορισμένες τροποποιήσεις. Συγκεκριμένα, έχουν αφαιρεθεί όλοι οι αντίστροφοι ρόλοι, ενώ οι functional ρόλοι αντιμετωπίζονται ως απλοί. Χρησιμοποιούνται δύο εκδόσεις αυτής της οντολογίας:

1. Η **NOT-GALEN**, μία περιορισμένη έκδοση που διαθέτει 4.379 αξιώματα, τα οποία αναφέρονται σε 2.748 ονοματικές έννοιες και 413 ονοματικούς ρόλους και
2. η **FULL-GALEN**, η οποία είναι και ίδια με την πλήρη GALEN αν εξαιρέσουμε τους αντίστροφους και functional ρόλους. Η οντολογία αυτή διαθέτει 38.093 αξιώματα τα οποία αναφέρονται σε 23.141 ονοματικές έννοιες και 950 ονοματικούς ρόλους.

9.2.2 Θησαυρός National Cancer Institute

Η οντολογία αυτή παρέχεται από το Εθνικό Ινστιτούτο Καρκίνου των Η.Π.Α (National Cancer Institute of the United States). Εκδίδεται σε μηνιαία βάση και αποτελεί μια ορολογία αναφοράς και βιοϊατρική οντολογία η οποία χρησιμοποιείται από το Ινστιτούτο και τους συνεργάτες του. Καλύπτει όρους κλινικής φροντίδας, μετάφρασης, βασικής έρευνας, δημοσιοποίησης πληροφοριών και διοικητικών λειτουργιών. Ο Θησαυρός του NCI παρέχει ορισμούς, συνωνυμίες και άλλες πληροφορίες, καθώς και περίπου 10.000 είδη καρκίνου και σχετιζόμενων ασθενειών, 8.000 μεμονωμένους δραστικούς παράγοντες και συνδυασμένες θεραπείες. Τέλος, περιέχεται και ένα ευρύ σύνολο άλλων θεμάτων σχετικών με τον καρκίνο και τη βιοϊατρική έρευνα. Η οντολογία αυτή συντηρείται από μία ομάδα ειδικών προερχόμενων από διαφορετικούς επιστημονικούς χώρους, οι οποίοι εισάγουν κατά μέσο όρο 900 νέες καταχωρίσεις κάθε μήνα. ([28])

Παρ' όλα αυτά, η δομή της οντολογίας είναι σχετικά απλή και είναι δυνατό να ταξινομηθεί από μηχανές συλλογιστικής που υποστηρίζουν την $\mathcal{EL}^+$.

9.2.3 Gene Ontology

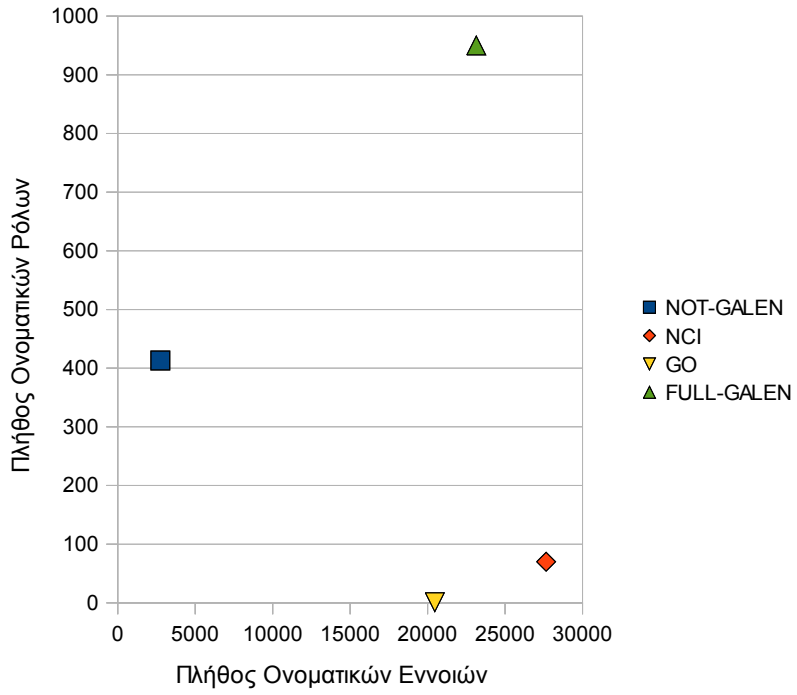
Η οντολογία αυτή είναι μία πολύ σημαντική προσπάθεια στο χώρο της βιοπληροφορικής που στοχεύει στην τυποποίηση της αναπαράστασης των γονιδίων και των χαρακτηριστικών της έκφρασής τους μεταξύ των διαφορετικών ειδών οργανισμών και των διαφορετικών βάσεων δεδομένων. Έτσι, παρέχεται ένα ελεγχόμενο σύνολο ορολογίας για την περιγραφή των χαρακτηριστικών έκφρασης των γονιδίων καθώς και την σύνταξη των σχετικών επισημειώσεων – περιγραφών και μεταδεδομένων από τους φορείς που συμμετέχουν στην προσπάθεια αυτή. ([29])

Στον επόμενο πίνακα θα συνοψισθούν τα βασικά ποσοτικά χαρακτηριστικά των οντολογιών που προαναφέρθηκαν:

	NOT-GALEN	**NCI Thesaurus**	**Gene Ontology**	**FULL-GALEN**
Πλήθος Αξιωμάτων	4.379	395.124	68.904	38.093
Πλήθος Ονοματικών Εννοιών	2.748	27.652	20.465	23.141
Πλήθος Ονοματικών Ρόλων	413	70	1	950
Εκφραστικότητα	ALEH+	ALE	ALE+	ALEH+

<u>Σημείωση:</u> Η εκφραστικότητα της κάθε οντολογίας παρέχεται από το OWL API. Η ύπαρξη των γραμμάτων "AL" στα ονόματα των κατηγοριών εκφραστικότητας δε θα πρέπει να συγχέεται από τον αναγνώστη με πιθανή ομοιότητα με την Περιγραφική Λογική $\mathcal{AL}$, της οποίας η εκφραστικότητα είναι διαφορετική από αυτή της μελετούμενης οικογένειας $\mathcal{EL}$.

Παρουσιάζονται επίσης εποπτικά οι συσχετίσεις μεταξύ των παραπάνω οντολογιών:



Παρατηρείται ότι οι δύο οντολογίες GALEN παρουσιάζουν αρκετά πιο ισόρροπη χρήση των ρόλων και εννοιών σε σύγκριση με τις άλλες δύο οντολογίες, οι οποίες είναι "φτωχές" σε ρόλους.

9.3 Πειραματικές Μετρήσεις

Κάθε μία από τις προαναφερόμενες οντολογίες ελήφθη από το [27] και έγινε προσπάθεια ταξινόμησής της, χρησιμοποιώντας τις τελευταίες εκδόσεις τριών βασικών μηχανών συλλογιστικής, καθώς και την υλοποιηθείσα μηχανή συλλογιστικής, η οποία στο εξής θα καλείται "f-EL$^+$ Reasoner". Συγκεκριμένα, έγινε χρήση των ακολούθων μηχανών:

1. CEL 1.0 build 8 ([21])
2. FaCT++ 1.3.0 ([22])
3. Pellet 2.0.0 rc7 ([30]) και
4. του υλοποιηθέντος f-EL$^+$ Reasoner

Οι δοκιμές έγιναν σε υπολογιστή με επεξεργαστή Intel Core2 Duo T7500 και 2GB RAM. Το λειτουργικό σύστημα ήταν ο πυρήνας Linux 2.6.28-13. Επίσης, χρησιμοποιήθηκε η έκδοση 6.14 της εικονικής μηχανής Java (JVM) της Sun. Όλες οι εκτελέσεις έγιναν σε βασικό περιβάλλον κειμένου, είχαν δηλαδή εκκινηθεί μόνο τα απαραίτητα προγράμματα και υπηρεσίες (χωρίς γραφικό περιβάλλον εργασίας). Η επιλογή αυτή έγινε προκειμένου να είναι όσο το δυνατόν πιο αξιόπιστες οι τελικές μετρήσεις και να μην επηρεαστούν αυτές από άλλες εφαρμογές που μπορεί να εκτελούνται την ίδια στιγμή. Η μόνη μηχανή συλλογιστικής για την οποία δεν ακολουθήθηκε αυτή η διαδικασία ήταν ο FaCT++, καθώς για τη λειτουργία του ήταν απαραίτητη η χρήση του Protégé. Αυτό συνέβη γιατί οι οντολογίες που χρησιμοποιήθηκαν δεν παρέχονται σε αναγνωριζόμενη από τον FaCT++ μορφή εισόδου, ενώ το παρεχόμενο εργαλείο μετατροπής ([31]) δε

μπορούσε να χειριστεί οντολογίες τέτοιου μεγέθους. Σε κάθε διεργασία προγράμματος συλλογιστικής δόθηκε η μέγιστη δυνατή προτεραιότητα.

Για την υλοποιηθείσα μηχανή συλλογιστικής κατά την εκτέλεση της σειράς μετρήσεων για σαφείς οντολογίες χρησιμοποιήθηκε το όρισμα "-nf", προκειμένου να μη λάβει χώρα ο έλεγχος που κανονικά διενεργείται κατά το στάδιο του ελέγχου εκφραστικότητας για ύπαρξη ασαφών αξιωμάτων υπαγωγής. Η απενεργοποίηση αυτή έγινε γιατί ο έλεγχος εισάγει μία μικρή καθυστέρηση στο αντίστοιχο στάδιο επεξεργασίας της μηχανής συλλογιστικής – ιδίως σε οντολογίες με μεγάλο πλήθος επισημειώσεων, όπως η NCI και η Gene Ontology. Αν ο έλεγχος αυτός ήταν ενεργοποιημένος, αυτό θα σήμαινε ότι οι μηχανές συλλογιστικής δε θα είχαν ελεγχθεί υπό τις ίδιες συνθήκες. Παρ' όλα αυτά, τα αποτελέσματα ποιοτικά θα παρέμεναν ίδια, όπως το ίδιο θα συνέβαινε και με τη σχετική κατάταξη των μηχανών συλλογιστικής, βάσει της ταχύτητας ταξινόμησης για κάθε οντολογία.

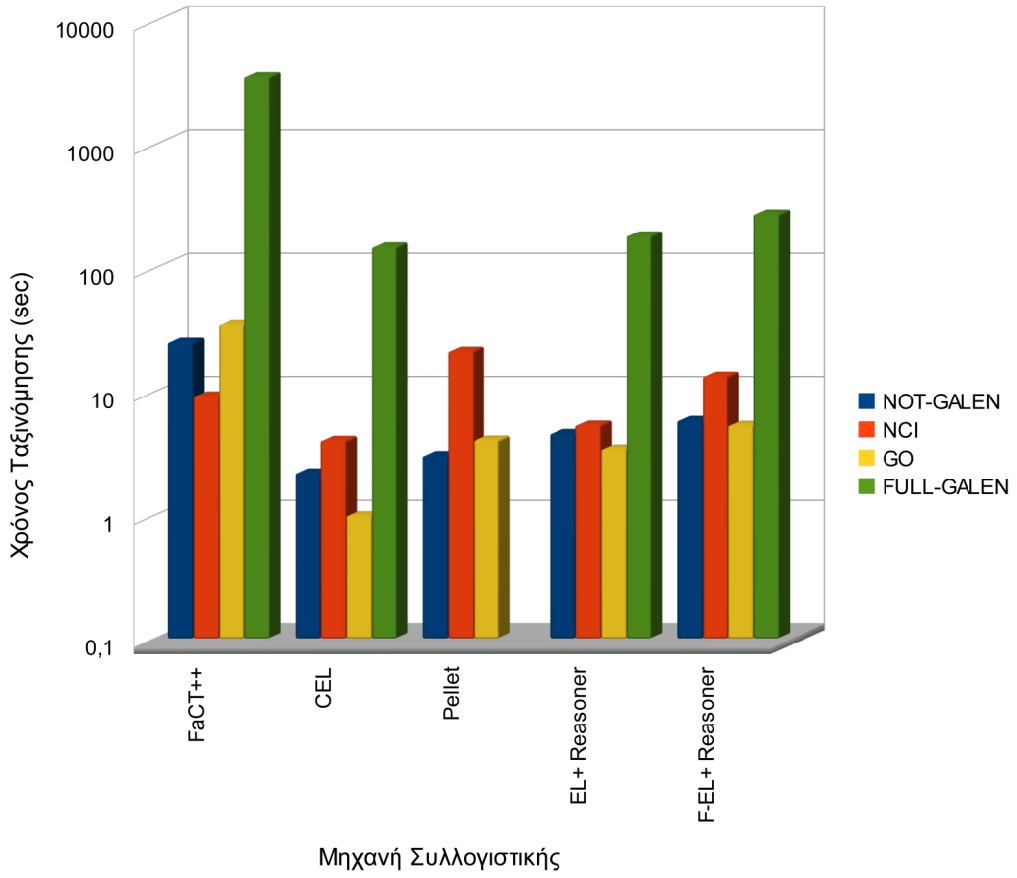
Επίσης, από την υλοποιηθείσα μηχανή συλλογιστικής f-EL$^+$ Reasoner και τον CEL ζητήθηκε και ο υπολογισμός των άμεσων σχέσεων υπαγωγής, γεγονός που αυξάνει σε μικρό βαθμό τον χρόνο εκτέλεσης..

9.3.1 Σαφής Αλγόριθμος Συλλογιστικής

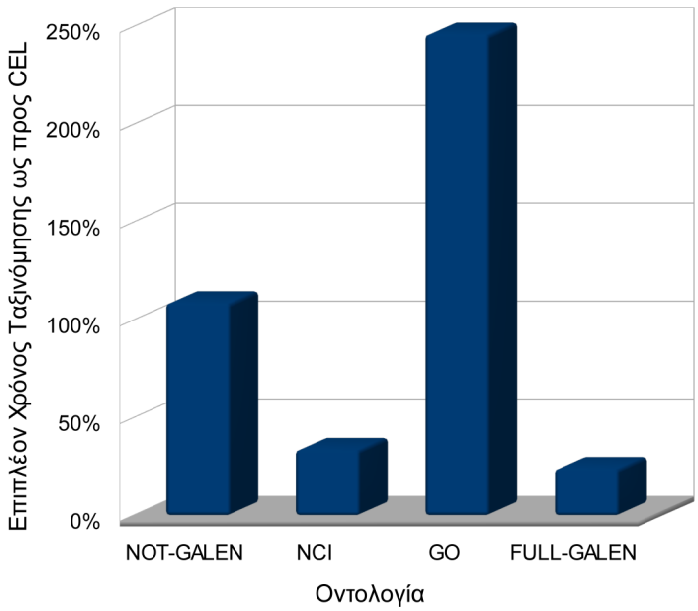
Παρακάτω παρουσιάζεται ο χρόνος ταξινόμησης για κάθε μία από τις τέσσερις βασικές σαφείς οντολογίες σε δευτερόλεπτα:

	NOT-GALEN	**NCI**	**GO**	**FULL-GALEN**
Χρόνος FaCT++ 1.3.0	24,88	9,11	34,42	3508,47
Χρόνος CEL 1.0 build 8	2,18	3,98	0,98	146,92
Χρόνος Pellet 2.0.0 rc7	3,00	21,00	4,00	Εσωτερικό Σφάλμα
f-EL$^+$ Reasoner	4,54	5,30	3,39	181,44
Λόγος (f-EL$^+$ Reasoner) / CEL	2,08	1,33	3,46	1,23

Τα αποτελέσματα αυτά, μαζί με τον χρόνο ταξινόμησης ασαφούς οντολογίας, παρουσιάζονται στο ακόλουθο διάγραμμα, όπου ως EL$^+$ Reasoner καλείται ο σαφής αλγόριθμος της παρούσας μηχανής συλλογιστικής και ως f-EL$^+$ Reasoner ο ασαφής:



Στο επόμενο διάγραμμα παρουσιάζεται ο επιπλέον χρόνος που απαιτείται για την ταξινόμηση κάθε σαφούς οντολογίας σε σχέση με τον χρόνο που απαιτεί για την ίδια εργασία ο CEL:



9.3.1.1 Σχολιασμός Μετρήσεων Σαφούς Αλγορίθμου

Από τις παραπάνω μετρήσεις είναι δυνατόν να εξαχθούν ορισμένα πολύ χρήσιμα συμπεράσματα. Κατ' αρχάς, γίνεται αντιληπτό πως τις καλύτερες επιδόσεις στην ταχύτητα ταξινόμησης τις παρουσιάζει ο CEL. Αυτό ήταν αναμενόμενο, κατά πρώτον λόγω της γλώσσας προγραμματισμού που έχει υλοποιηθεί (Lisp), η οποία και είναι πολύ ταχύτερη από τη γλώσσα Java, στην οποία υλοποιείται ο Pellet και ο υλοποιηθείς στα πλαίσια της παρούσας εργασίας f-EL$^+$ Reasoner. Κατά δεύτερον, η έκδοση του CEL που δοκιμάστηκε είναι μία ώριμη έκδοση αυτού, αποτέλεσμα εργασιών ανάπτυξης και βελτιώσεων περίπου τεσσάρων ετών. Ο CEL παρουσιάστηκε για πρώτη φορά το έτος 2005 και οι αρχικές εκδόσεις αυτού δεν επιτύγχαναν τόσο καλές επιδόσεις ([15]).

Από τα υπόλοιπα συστήματα συλλογιστικής, το δεύτερο σε επιδόσεις είναι ο f-EL$^+$ Reasoner. Είναι το δεύτερο ταχύτερο στην ταξινόμηση τριών από τις τέσσερις οντολογίες σύγκρισης, ενώ ειδικά για την μεγαλύτερη και πολυπλοκότερη όλων, τη FULL-GALEN, απαιτεί μόνο 23% περισσότερο χρόνο για την ταξινόμησή της σε σχέση με τον CEL. Ακόμη, είναι η μόνη μηχανή συλλογιστικής από τις υπόλοιπες που καταφέρνει τελικά να ταξινομήσει την οντολογία αυτή, καθώς ο μεν FaCT++ απαιτεί σχεδόν μία ώρα, χρονικό διάστημα μη αποδεκτό και μη συγκρίσιμο με τις υπόλοιπες μηχανές συλλογιστικής, ενώ ο δε Pellet τερματίζει τη λειτουργία του λόγω κάποιου εσωτερικού σφάλματος. Επίσης, ακόμα και σε οντολογίες με μεγάλο πλήθος αξιωμάτων και ονοματικών εννοιών όπως η NCI, ο f-EL$^+$ Reasoner απαιτεί μόνο 33% περισσότερο χρόνο για την ταξινόμηση σε σχέση με τον ταχύτερο CEL. Ταυτόχρονα, στη σχετικά μικρή οντολογία NOT-GALEN, η οποία παρουσιάζει ισορροπημένη χρήση των εκφραστικών μέσων της $\mathcal{EL}^+$, ο χρόνος που απαιτείται για την ταξινόμηση είναι περίπου ο διπλός σε σύγκριση με τον CEL.

Όσον αφορά τον Pellet, σημαντικό είναι να αναφερθεί πως οι επιδόσεις του οφείλονται στον εξειδικευμένο αλγόριθμο χειρισμού $\mathcal{EL}$ οντολογιών που υλοποιεί, παρά το γεγονός ότι εν γένει μπορεί να υποστηρίξει υπηρεσίες συλλογιστικής σε Περιγραφικές Λογικές υψηλότερης εκφραστικότητας. Κατά μέσο όρο παρουσιάζει ικανοποιητικά αποτελέσματα, όπως διακρίνεται και από τις επιδόσεις του στην ταξινόμηση των οντολογιών NOT-GALEN και GO (Gene Ontology). Για απροσδιόριστο λόγο παρουσιάζει τις χειρότερες επιδόσεις από όλες τις μηχανές συλλογιστικής στην ταξινόμηση της οντολογίας NCI – πράγμα το οποίο δείχνει ότι παρά την απλή δομή της οντολογίας, η ταχύτητά του περιορίζεται σε πολύ σημαντικό βαθμό από το μέγεθός της (πλήθος αξιωμάτων και ονοματικών εννοιών). Ένα πολύ ενδιαφέρον σημείο θα ήταν οι επιδόσεις του στην ταξινόμηση της FULL-GALEN, οι οποίες όμως στάθηκε αδύνατο να ληφθούν καθώς εμφάνιζε ένα εσωτερικό σφάλμα.

Τέλος, τα αποτελέσματα χρόνου εκτέλεσης για τον FaCT++ καταδεικνύουν την υψηλή χρονική πολυπλοκότητα του αλγορίθμου συλλογιστικής που υλοποιεί, γεγονός αναμενόμενο εξ' αιτίας της υψηλής εκφραστικότητας που υποστηρίζει. Αυτό έχει σαν αποτέλεσμα να παρουσιάζει εν γένει τις χειρότερες επιδόσεις από τις τέσσερις μηχανές συλλογιστικής και μόνο στην απλής δομής οντολογία NCI καταφέρνει να καταλάβει την τρίτη θέση, μπροστά από τον Pellet. Βέβαια, το πολύ μεγάλο πλήθος εννοιών και αξιωμάτων της οντολογίας αυτής αναδεικνύει και την αποτελεσματικότητα των ευρετικών μεθόδων που αυτός υλοποιεί ([23]), οι οποίες σε πολλές καθημερινές περιπτώσεις του εξασφαλίζουν πολύ καλή ταχύτητα συλλογιστικής παρά την υψηλή πολυπλοκότητα του υλοποιούμενου αλγορίθμου.

Ως συμπέρασμα, με την υλοποίηση στα πλαίσια της παρούσας διπλωματικής εργασίας της μηχανής συλλογιστικής για την Περιγραφική Λογική $f-\mathcal{EL}^+$ αναδεικνύεται κατ' αρχάς η σημασία της ύπαρξης ενός αλγορίθμου συλλογιστικής χαμηλής χρονικής πολυπλοκότητας, δη-

λαδή πολυωνυμικής πολυπλοκότητας χαμηλού βαθμού ή καλύτερης. Και οι τρεις μηχανές που επιτυγχάνουν τις καλύτερες επιδόσεις υλοποιούν εξειδικευμένους πολυωνυμικής πολυπλοκότητας αλγορίθμους συλλογιστικής. Κατά δεύτερον, το χάσμα επιδόσεων μεταξύ ενός προγράμματος υλοποιημένου σε μία κατά γενική ομολογία γρήγορη γλώσσα προγραμματισμού – όπως η Lisp – και ενός προγράμματος υλοποιημένου σε μία γλώσσα προγραμματισμού που θεωρείται βραδύτερη – όπως η Java – είναι πολύ μικρότερο απ' ότι αναμενόταν. Αυτό οφείλεται κατά κύριο λόγο στην ευκολία με την οποία σε μία γλώσσα όπως η Java μπορεί ο προγραμματιστής να ορίσει και να χρησιμοποιήσει πολύπλοκες και ταχύτατες βοηθητικές δομές δεδομένων, όπως HashMaps ή Caches. Τέτοιες πολύπλοκες δομές ενσωματώνονται πολύ εύκολα σε κάποιο πρόγραμμα, ενώ η χρήση τους στα κατάλληλα σημεία μπορεί να οδηγήσει σε σημαντικότατη αύξηση της ταχύτητάς του. Κατά τρίτον, μπορεί να παρατηρηθεί πως ο χρόνος συλλογιστικής εξαρτάται κατά κύριο λόγο από το πλήθος των ονοματικών ρόλων που εμφανίζονται στην οντολογία και από την πολυπλοκότητα των μεταξύ τους συσχετίσεων, ενώ μόνο δευτερευόντως επηρεάζεται ο χρόνος εκτέλεσης από το πλήθος των αξιωμάτων ή των ονοματικών εννοιών.

Ακόμα, μπορεί κάποιος να ισχυριστεί ότι ο υλοποιηθείς f-EL$^+$ Reasoner εμφανίζει "καλύτερη συμπεριφορά" όσο αυξάνεται το πλήθος των ονοματικών ρόλων της οντολογίας, προσεγγίζοντας σε μεγαλύτερο βαθμό την απόδοση του ταχύτερου CEL. Το γεγονός αυτό γίνεται αντιληπτό αν συγκριθεί η διαφορά επίδοσης με τον CEL για την ταξινόμηση της οντολογίας GO σε σχέση με τις υπόλοιπες: για την ταξινόμηση της οντολογίας αυτής απαιτείται περίπου 3,5 φορές ο χρόνος στον οποίο την ταξινομεί ο CEL, ενώ για τις άλλες οντολογίες που περιέχουν πολλούς περισσότερους ρόλους, ο λόγος αυτός μειώνεται και φτάνει το 1,23 για την περίπτωση της FULL-GALEN. Γενικά, ο f-EL$^+$ Reasoner δείχνει να παρουσιάζει πολύ καλή κλιμάκωση (scaling) των επιδόσεων για αυξανόμενο πλήθος ρόλων σε σύγκριση με την αντίστοιχη κλιμάκωση που παρουσιάζει ο CEL. Αυτό οφείλεται στη σημαντική προσπάθεια που καταβλήθηκε για τη βελτιστοποίηση των δομών αναπαράστασης των συνόλων ρόλων (role sets) R, καθώς και στη σχετική κρυφή μνήμη (cache) που εισήχθη. Αντιθέτως, για τον χειρισμό των αξιωμάτων και των εννοιών χρησιμοποιούνται οι παρεχόμενες από το OWL API δομές, ενώ δεν εισήχθη κάποια βοηθητική δομή βελτιστοποίησης.

Τέλος, πολύ σημαντικό πλεονέκτημα των μηχανών συλλογιστικής που βασίζονται στο OWL API,όπως του f-EL$^+$ Reasoner και του Pellet, είναι η δυνατότητα χρήσης οντολογιών οι οποίες βρίσκονται καταγεγραμμένες σε μορφή OWL, χωρίς να απαιτείται προηγουμένως επεξεργασία σε κάποια ειδική μορφή σύνταξης, όπως συμβαίνει με τον CEL και τον FaCT++.

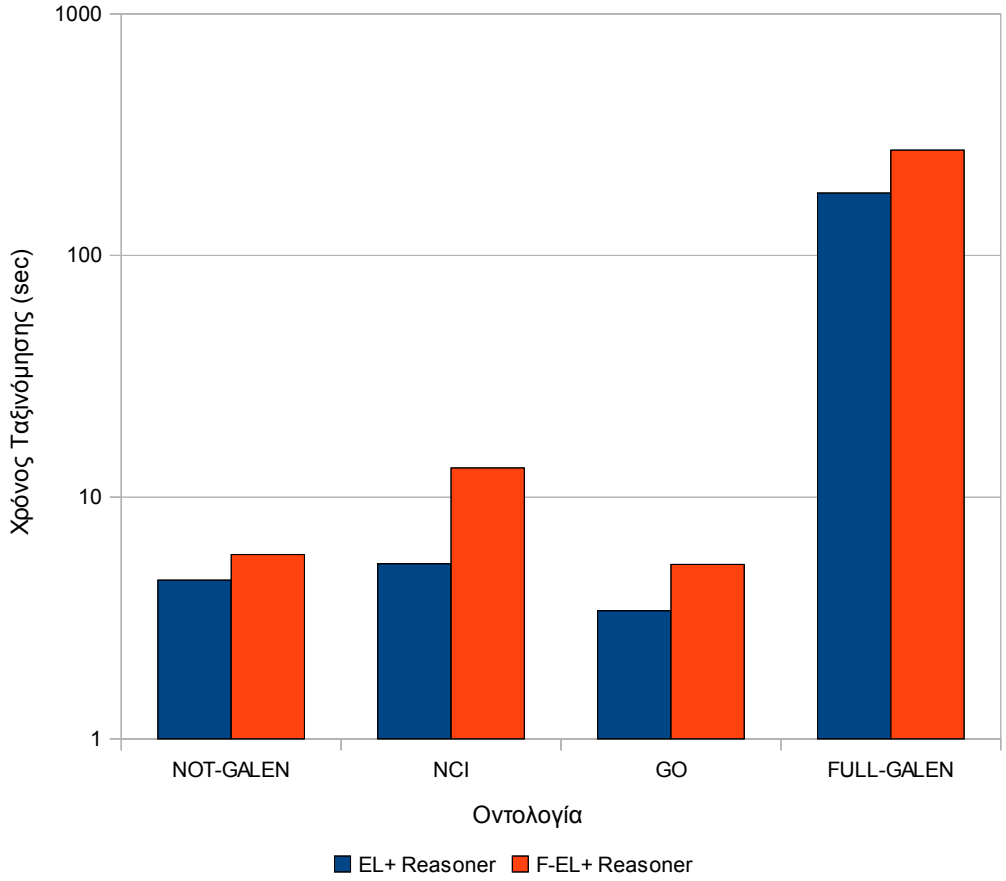
9.3.2 Ασαφής Αλγόριθμος Συλλογιστικής

Για την λήψη των παρακάτω μετρήσεων του απαιτούμενου χρόνου ταξινόμησης χρησιμοποιήθηκαν ασαφείς εκδόσεις των ανωτέρω οντολογιών, οι οποίες οδηγούσαν το πρόγραμμα στη χρήση του ασαφούς αλγορίθμου συλλογιστικής. Ο χρόνος εκτέλεσης απεικονίζεται σε δευτερόλεπτα.

	Fuzzified NOT-GALEN	Fuzzified NCI	Fuzzified GO	Fuzzified FULL-GALEN
Χρόνος f-EL$^+$ Reasoner	5,73	13,19	5,32	272,94*
Επιπλέον χρόνος σε σχέση με σαφή οντολογία	26%	149%	57%	50%*

Σημείωση: Κατά τις μετρήσεις για την ασαφοποιημένη οντολογία FULL-GALEN, παρατηρήθηκε ελαφρά χρήση του swap file, γεγονός το οποίο σημαίνει ότι ο πραγματικός χρόνος εκτέλεσης αν υπήρχε περισσότερη διαθέσιμη μνήμη θα ήταν μικρότερος. Η χρήση του swap file δεν αντιστοιχεί σε κατανάλωση μνήμης του ιδίου του προγράμματος, καθώς είχε γίνει ειδική ρύθμιση ώστε η μνήμη που του αναλογεί να μην ξεπεράσει σε καμία περίπτωση τη διαθέσιμη φυσική μνήμη του συστήματος (όρισμα -Xmx στην εικονική μηχανή Java). Κρίνεται λοιπόν πως το φαινόμενο οφείλεται σε ελλιπή λειτουργία του συλλέκτη σκουπιδιών (garbage collector) της Java, όπως παρατηρήθηκε από τα σχετικά εργαλεία που κατέγραφαν την κατάσταση των πόρων του υπολογιστή.

Στο παρακάτω διάγραμμα απεικονίζεται η διαφορά χρόνου εκτέλεσης για την ταξινόμηση της σαφούς και της ασαφοποιημένης έκδοσης κάθε μίας από τις τέσσερις οντολογίες:



9.3.2.1 Σχολιασμός Μετρήσεων Ασαφούς Αλγορίθμου

Παρατηρούμε πως ο απαιτούμενος χρόνος για την ταξινόμηση ασαφούς οντολογίας είναι εν γένει μεγαλύτερος του χρόνου ταξινόμησης της αντίστοιχης σαφούς οντολογίας. Η διαφορά αυτή είναι αναμενόμενη και οφείλεται τόσο στις πολυπλοκότερες δομές που πρέπει να χρησιμοποιεί και να διατηρεί στη μνήμη ο ασαφής αλγόριθμος, όσο και στον έλεγχο που γίνεται έτσι ώστε να εισάγονται πάντα εγγραφές στις ουρές και στα σύνολα με τον κατάλληλο βαθμό βεβαιότητας. Έτσι, σε οντολογίες με καλή ισορροπία ανάμεσα στο πλήθος των ονοματικών εννοιών και των ρόλων, όπως οι δύο εκδόσεις της GALEN, παρατηρείτε μία αύξηση του χρόνου εκτέλεσης η οποία γενικά κυμαίνεται μεταξύ του 25% έως 50% περίπου του χρόνου ταξινόμησης της αντίστοιχης οντολογίας με τον σαφή αλγόριθμο. Σε άλλες οντολογίες, οι οποίες έχουν σημαντικά μεγαλύτερο πλήθος αξιωμάτων καθώς και λιγότερους έως ελάχιστους επώνυμους ρόλους καθώς και απλούστερη δομή, η καθυστέρηση που εισάγει ο ασαφής αλγόριθμος ως ποσοστό του χρόνου σαφούς ταξινόμησης είναι μεγαλύτερη, και κυμαίνεται εν γένει από 50% έως 150% του αρχικού χρόνου εκτέλεσης.

Αυτή η "αποδοτικότερη λειτουργία" που παρουσιάζει η υλοποιηθείσα μηχανή συλλογιστικής για οντολογίες με πολλούς ονοματικούς ρόλους έχει παρατηρηθεί και σχολιαστεί και στην ενότητα του σαφούς αλγορίθμου.

9.4 Συμπεράσματα και Προτάσεις

Γενικά, κατά την υλοποίηση της μηχανής συλλογιστικής έγινε αντιληπτή η σημασία της προσπάθειας βελτιστοποίησης ενός αλγορίθμου στην τελική απόδοση του προγράμματος. Συγκεκριμένα, έχει ήδη αναφερθεί σε προηγούμενο κεφάλαιο πως, αν και οι αλγόριθμοι που υλοποιήθηκαν τόσο για το τμήμα σαφούς όσο και για το τμήμα ασαφούς συλλογιστικής είναι πολυωνυμικοί, οι πρώτες εκδόσεις αυτών ήταν περίπου δύο τάξεις μεγέθους βραδύτερες από τις τελικές. Το γεγονός αυτό καταδεικνύει πως ακόμα και σε Περιγραφικές Λογικές χαμηλής πολυπλοκότητας, όπως η οικογένεια των Περιγραφικών Λογικών $\mathcal{EL}$ ή η Περιγραφική Λογική DL-Lite, δεν αρκεί η ύπαρξη ενός "βατού" (πολυωνυμικού) αλγορίθμου: καίρια είναι η συμβολή της υλοποίησής του στην τελική ταχύτητα που θα παρουσιάζει το σύστημα. Σε διαφορετική περίπτωση, είναι δυνατόν για πολλές καθημερινές περιπτώσεις ένας ιδιαίτερα βελτιστοποιημένος μεν, υψηλότερης πολυπλοκότητας δε αλγόριθμος, όπως αυτός που υλοποιείται από τον FaCT++, να καταλήγει να είναι ταχύτερος.

Στη συνέχεια θα αναφερθούν ορισμένες προτάσεις, οι οποίες καθορίζουν τα βασικά σημεία στα οποία κρίνεται σκόπιμο να δοθεί έμφαση κατά τη διάρκεια μίας πιθανής μελλοντικής επέκτασης της παρούσας εργασίας, είτε στα πλαίσια του συστήματος FReS (Fuzzy Reasoning Services), είτε αυτόνομης.

1. Αρχικά, ιδιαίτερο βάρος θα πρέπει να δοθεί στην **προσπάθεια περαιτέρω επιτάχυνσης του αλγορίθμου**. Η τρέχουσα υλοποίηση μπορεί να ειπωθεί πως παρουσιάζει καλύτερη συμπεριφορά για αυξανόμενο πλήθος ονοματικών ρόλων σε μία οντολογία, παρ' ότι για αυξανόμενο πλήθος αξιωμάτων ή ονοματικών εννοιών. Με τον όρο "καλύτερη συμπεριφορά", εννοείται η καλύτερη απόδοση συγκριτικά με με την ταχύτερη μηχανή συλλογιστικής, τον CEL, για οντολογίες που παρουσιάζουν μεγάλο πλήθος ρόλων σε σχέση με αυτές που χρησιμοποιούν λιγότερους ρόλους. Βέβαια, η καλύτερη συμπεριφορά αυτή οφείλεται στο γεγονός ότι εκεί δόθηκε και το κύριο βάρος της υλοποίησης, δηλαδή στην επιτάχυνση του χειρισμού των ρόλων. Η απόφαση αυτή ελήφθη καθώς ήδη από τα αρχι-

κά στάδια της υλοποίησης διαπιστώθηκε ότι η οριακή επιβάρυνση ενός νέου ονοματικού ρόλου είναι πολύ υψηλότερη της οριακής επιβάρυνσης που επιφέρει ένα νέο αξίωμα ή μία νέα έννοια. Απλουστευμένα, δε θα ήταν λανθασμένη η θεώρηση ότι η πολυπλοκότητα της συλλογιστικής για τον αλγόριθμο αυτόν εξαρτάται κατά κύριο λόγο από το πλήθος των εμφανιζόμενων ρόλων και των μεταξύ αυτών συσχετίσεων, ενώ το πλήθος των αξιωμάτων και των εννοιών συνεισφέρει λιγότερο στην αύξηση του απαιτούμενου χρόνου εκτέλεσης.

Έτσι, θα ήταν σημαντικό να διερευνηθεί κατά πόσον είναι εφικτή η επιτάχυνση των τμημάτων εκείνων του αλγορίθμου που διαχειρίζονται τα αξιώματα και τις έννοιες. Η επιτάχυνση αυτή θα μπορούσε να επιτευχθεί με τρεις τρόπους:

i. **Με την αντικατάσταση της έκδοσης του OWL API που χρησιμοποιείται με κάποια νεότερη.** Η έκδοση του OWL API η οποία χρησιμοποιείται από τον αλγόριθμο συλλογιστικής βασίζεται στην 2η έκδοσή του. Δε χρησιμοποιήθηκε η τελευταία διατιθέμενη έκδοση 2.2.0, αλλά μία νεότερη έκδοση η οποία επιδιορθώνει πολλά προβλήματα της βιβλιοθήκης, τα οποία εμφανίζονταν ιδίως στον χειρισμό των επισημειώσεων. Η έκδοση αυτή ήταν προσβάσιμη μέσω του SVN αποθετηρίου κώδικα (Subversion Repository) που χρησιμοποιούν οι κατασκευαστές της βιβλιοθήκης για την οργάνωση της ανάπτυξής της.

   Κατά τον χρόνο ολοκλήρωσης της παρούσας διπλωματικής εργασίας – Ιούλιος 2009 –, η επόμενη, 3η, έκδοση του OWL API βρισκόταν ακόμη σε αρχικό στάδιο ανάπτυξης, γι' αυτό και δε χρησιμοποιήθηκε. Παρ' όλα αυτά, ο κώδικας έχει σχεδιαστεί κατά τέτοιον τρόπο ώστε να κάνει όσο το δυνατόν μεγαλύτερη χρήση της βιβλιοθήκης και να διευκολύνει τη μετάβαση σε ένα νέο API. Έτσι, σε περίπτωση που αυτή περιλαμβάνει βελτιώσεις στα κρίσιμα σημεία που απαιτεί ο κώδικας της μηχανής συλλογιστικής, αυτό θα σημάνει μία επιτάχυνση της ταχύτητας ταξινόμησης οντολογιών.

ii. Ένας δεύτερος τρόπος θα ήταν με τη διερεύνηση της σκοπιμότητας **εισαγωγής νέων βοηθητικών δομών** διαχείρισης αξιωμάτων και εννοιών, όπως οι κρυφές μνήμες ή τα σύνολα απεικονίσεων HashMaps, καθώς και νέων τρόπων αναπαράστασης των ήδη χρησιμοποιούμενων δομών στη μνήμη. Δεν είναι απίθανο λύσεις που βοήθησαν στην επιτάχυνση του αλγορίθμου να μπορούν να επαναχρησιμοποιηθούν με όμοιο τρόπο και σε νέα σημεία του προγράμματος, αυξάνοντας ακόμα περισσότερο την ταχύτητά του. Επίσης, μία μέθοδος η οποία βοήθησε στην επιτάχυνση της εκτέλεσης του αλγορίθμου σε συγκεκριμένα σημεία ήταν αυτή της αλλαγής της σειράς με την οποία βρίσκονται εμφωλευμένοι οι βρόχοι επανάληψης. Η μέθοδος αυτή ίσως να μπορούσε να χρησιμοποιηθεί και σε μεγαλύτερο βαθμό.

   Ένα σημαντικό σημείο που χρήζει προσπάθειας επιτάχυνσης είναι το στάδιο στο οποίο ο ασαφής αλγόριθμος συλλογιστικής υπολογίζει τα σύνολα αντιστοιχίσεων και ιδίως τις εγγραφές που πρέπει να εισαχθούν κάθε φορά σε κάποια από τις ουρές.

iii. Ένα τρίτο ερώτημα το οποίο θα πρέπει να διερευνηθεί είναι εάν και κατά πόσον μπορούν να γίνουν περαιτέρω **βελτιώσεις στον θεωρητικό αλγόριθμο συλλογιστικής**, μειώνοντας και άλλο την θεωρητική χρονική του πολυπλοκότητα.

2. Μία δεύτερη κατεύθυνση στην οποία θα πρέπει να δοθεί ιδιαίτερο βάρος είναι αυτή της **παραλληλοποίησης του αλγορίθμου**. Η σύγχρονη τάση διάδοσης κεντρικών μονάδων επεξεργασίας που αποτελούνται από πολλαπλούς πυρήνες σε συνδυασμό με τη μείωση του ρυθμού αύξησης της απόδοσης που προσφέρει ο κάθε νέος μεμονωμένος πυρήνας σε

σύγκριση με παλαιότερα, οδηγούν σε αδιέξοδο την παραδοσιακή υλοποίηση σειριακών αλγορίθμων. Το πρόβλημα αυτό είναι ιδιαίτερα φανερό σε περιβάλλοντα όπου η ικανότητα ταξινόμησης οντολογιών του υπολογιστικού συστήματος με χρήση κάποιου σειριακού αλγορίθμου υπερβαίνει τη ζήτηση των χρηστών για ταξινομήσεις οντολογιών. Απλούστερα, σε περιβάλλοντα όπου ο αριθμός των μη ανταγωνιζόμενων αλγορίθμων που μπορούν να εκτελούνται παράλληλα υπερβαίνει τον αριθμό των οντολογιών που χρειάζεται να ταξινομούνται ταυτόχρονα. Σε ένα τέτοιο περιβάλλον, θα παρουσιάζεται χαμηλή χρησιμοποίηση (utilisation) των υπολογιστικών πόρων χωρίς ο χρήστης να μπορεί να ωφεληθεί από αυτούς. Με την πάροδο των ετών θα συναντώνται όλο και συχνότερα τέτοιες περιπτώσεις, των οποίων οι επιπτώσεις δε θα είναι πάντα δυνατό ή επιθυμητό να περιοριστούν με χρήση εικονικών μηχανών (virtualisation) – δηλαδή με υλοποίηση πολλών "αδύναμων" εικονικών υπολογιστών εντός ενός πραγματικού "ισχυρότερου".

Οι εξελίξεις αυτές έχουν σημαντικό αντίκτυπο στους αλγορίθμους συλλογιστικής, ιδίως με την ταχύτατη διάδοση της χρήσης τους και των σχετικών τεχνολογιών. Επομένως, η παραλληλοποίηση αυτών κρίνεται αναγκαία τόσο για τις ήδη υπάρχουσες όσο και για τις μελλοντικές υλοποιήσεις. Στον αλγόριθμο που υλοποιήθηκε για την παρούσα διπλωματική εργασία, δύο είναι τα κύρια σημεία στα οποία μπορεί να εστιασθεί η προσπάθεια παραλληλοποίησης:

i. Στον διαχωρισμό των λειτουργιών του κυρίου νήματος εκτέλεσης σε δύο κατηγορίες: πρώτον, στην κατηγορία των βασικών ενεργειών του αλγορίθμου, οι οποίες και μπορεί να εκτελούνται ακολουθιακά και δεύτερον, σε ένα σύνολο βοηθητικών λειτουργιών, οι οποίες μπορούν να εκτελούνται από βοηθητικά νήματα (helper threads) σε περιβάλλον κοινής (μοιραζόμενης) μνήμης ή από ξεχωριστούς υπολογιστές σε περιβάλλον ξεχωριστής μνήμης. Το κύριο νήμα θα παράγει ένα σύνολο εντολών για τα βοηθητικά νήματα, όπως για παράδειγμα εντολές εισαγωγής στοιχείων σε σύνολα, υπολογισμού και ανάκτησης πληροφοριών ή συνόλων και άλλες. Μπορούν να ακολουθηθούν δύο μοντέλα ανάπτυξης: αυτό της επικοινωνίας των δύο τμημάτων με χρήση κάποιας ενδιάμεσης μνήμης (buffer), η οποία θα αποθηκεύει τις εντολές προς εκτέλεση και αυτό της υλοποίησης κάποιας συναλλακτικής (transactional) αρχιτεκτονικής παραλληλοποίησης.

ii. Το δεύτερο σημείο στο οποίο θα μπορούσαν να εστιαστούν μελλοντικές επεκτάσεις είναι αυτό της εγγενούς παραλληλοποίησης του αλγορίθμου συλλογιστικής. Η προσπάθεια παραλληλοποίησης αλγορίθμων συλλογιστικής δεν είναι κάτι καινούριο ([32]), αλλά σίγουρα είναι μία τάση η οποία αναμένεται να ενταθεί σημαντικά τα επόμενα χρόνια ([33]). Σε κάθε περίπτωση, στη βιβλιογραφία υπάρχει διαθέσιμο αρκετό σχετικό υλικό.

Στην παρούσα υλοποίηση του f-EL$^+$ Reasoner υπάρχουν λειτουργίες που θα μπορούσαν εύκολα να επιταχυνθούν με **διαίρεση του προβλήματος σε υποπροβλήματα** και παράλληλη εκτέλεση αυτών. Ορισμένες από αυτές είναι η διόρθωση του βαθμού βεβαιότητας των εγγραφών ουράς – όποτε αυτή απαιτείται –, στάδια του υπολογισμού των απαιτούμενων εγγραφών ουράς και άλλες λειτουργίες του αλγορίθμου που εκτελούνται επαναληπτικά σε μεγάλο σύνολο δεδομένων και δεν απαιτούν μεταξύ τους επικοινωνία κατά τη διάρκεια των επαναλήψεων. Μία τέτοια υλοποίηση, συνδυασμένη με τις τεχνικές αποθήκευσης (caching) των αποτελεσμάτων θα μπορούσε να προσφέρει μία αύξηση της επίδοσης κατά τουλάχιστον 30% στις γενικές περιπτώσεις.

3. Ένας άλλος τομέας που θα μπορούσε να βελτιωθεί είναι αυτός της **δημιουργίας μίας τυποποιημένης μεθόδου διαπροσωπείας** για τον υλοποιηθέντα f-EL$^+$ Reasoner. Μία τέτοια διαπροσωπεία επικοινωνίας θα μπορούσε να είναι είτε μία διαπροσωπεία επικοινωνίας εφαρμογής-πελάτη και προγράμματος (Application Programming Interface, API), είτε μία διαπροσωπεία βασιζόμενη στο πρότυπο DIG (DL Implementation Group) ([24]). Η υλοποίηση της τελευταίας περίπτωσης δεν είναι ιδιαίτερα απλή, καθώς το πρότυπο DIG δεν καθορίζει κάποιο σχήμα μεταφοράς και επικοινωνίας για ασαφείς πληροφορίες, όπως οι βαθμοί βεβαιότητας.

   Η υλοποίηση κάποιας διαπροσωπείας μπορεί να γίνει είτε αυτόνομα για την παρούσα μηχανή συλλογιστικής, είτε ενιαία για το ολόκληρο το σύστημα FReS και όλες τις διαφορετικές μηχανές συλλογιστικής που αυτό θα υποστηρίζει.

4. Τέλος, πολύ σημαντική θα ήταν η επέκταση του υλοποιηθέντος f-EL$^+$ Reasoner, ώστε αυτός να υποστηρίζει τη **δυνατότητα αυξητικής ταξινόμησης (incremental classification)** και **διάσπασης σε υπομονάδες (modularisation)** της οντολογίας. Η ύπαρξη τέτοιων δυνατοτήτων θα ήταν πολύ σημαντική, ιδίως για αποδοτικότερη χρήση της μηχανής συλλογιστικής στα πλαίσια του συστήματος FReS (Fuzzy Reasoning Services) και την ταχύτερη ανταπόκριση στις αλλαγές που εισάγει ο χρήστης.

10. Βιβλιογραφία

[1] Γ. Στοΐλος, “Εισαγωγή στις Περιγραφικές Λογικές,” 2007.

[2] I.R. Horrocks, “Using an expressive description logic: FaCT or fiction?,” MORGAN KAUFMANN PUBLISHERS, 1998, pp. 636-649.

[3] R.L. de Mantaras and L. Saina, “Polynomial time reasoning in a description logic with existential restrictions, GCI axioms, and?what else?,” IOS Press, 2004, p. 298.

[4] A. Rector and I. Horrocks, “Experience building a large, re-usable medical ontology using a description logic with transitivity and concept inclusions,” 1997, pp. 321–325.

[5] M. Ashburner, C.A. Ball, J.A. Blake, D. Botstein, H. Butler, J.M. Cherry, A.P. Davis, K. Dolinski, S.S. Dwight, and J.T. Eppig, “Gene Ontology: tool for the unification of biology,” *Nature genetics*, vol. 25, 2000, pp. 25-29.

[6] B.C. Grau, I. Horrocks, B. Motik, B. Parsia, P. Patel-Schneider, and U. Sattler, “OWL 2: The next step for OWL,” *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 6, 2008, pp. 309-322.

[7] J. Mendez and B. Suntisrivaraporn, “Reintroducing CEL as an OWL 2 EL Reasoner.”

[8] F. Baader, S. Brandt, and C. Lutz, “Pushing the EL envelope,” LAWRENCE ERLBAUM ASSOCIATES LTD, 2005, p. 364.

[9] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati, “DL-Lite: Tractable description logics for ontologies,” Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2005, p. 602.

[10] T. Berners-Lee, J. Hendler, and O. Lassila, “The semantic web,” *Scientific American*, vol. 284, 2001, pp. 34-43.

[11] G. Stoilos, G. Stamou, V. Tzouvaras, J.Z. Pan, and I. Horrocks, “Fuzzy OWL: Uncertainty and the semantic web,” 2005.

[12] G. Stoilos, G. Stamou, and J.Z. Pan, “Classifying Fuzzy Subsumption in Fuzzy-EL+.”

[13] F. Baader, C. Lutz, and B. Suntisrivaraporn, “Efficient reasoning in EL+,” *Proceedings of DL2006*, vol. 189.

[14] F. Baader, C. Lutz, and B. Suntisrivaraporn, “Is tractable reasoning in extensions of the description logic EL useful in practice,” 2005.

[15] F. Baader, C. Lutz, and B. Suntisrivaraporn, “CEL-a polynomial-time reasoner for life science ontologies,” *Lecture Notes in Computer Science*, vol. 4130, 2006, p. 287.

[16] http://www.ihtsdo.org/, “IHTSDO: International Health Terminology Standards Development Organisation.”

[17] S. Bechhofer, R. Volz, and P. Lord, “Cooking the Semantic Web with the OWL API,” *Lecture Notes in Computer Science*, 2003, pp. 659-675.

[18] M. Horridge, S. Bechhofer, and O. Noppens, “Igniting the OWL 1.1 touch paper: The OWL API,” *Proc. OWL-ED*, vol. 258, 2007.

[19] http://owlapi.sourceforge.net/, “OWL API.”

[20] http://code.google.com/p/cel/, “cel - Google Code.”

[21] http://lat.inf.tu-dresden.de/systems/cel/, “CEL :: a polynomial-time Classifier for the description logic EL+.”

[22] http://code.google.com/p/factplusplus/, “factplusplus - Google Code.”

[23] D. Tsarkov and I. Horrocks, “FaCT++ description logic reasoner: System description,” *Lecture Notes in Computer Science*, vol. 4130, 2006, p. 292.

[24] http://dl.kr.org/dig/interface.html, “DL Implementation Group (DIG).”

[25] https://whirlycache.dev.java.net/, “Whirlycache.”

[26] B. Suntisrivaraporn, “Optimization and implementation of subsumption algorithms for the description logic EL with cyclic TBoxes and general concept inclusion axioms,” Master thesis, TU Dresden, Germany, 2005.

[27] http://lat.inf.tu-dresden.de/~meng/ontologies/, “Meng » Ontologies.”

[28] http://bioportal.nci.nih.gov/ncbo/faces/pages/ontology_list.xhtml, “The NCICB - BioPortal.”

[29] http://www.geneontology.org/index.shtml, “the Gene Ontology.”

[30] http://clarkparsia.com/pellet, “Pellet: The Open Source OWL DL Reasoner.”

[31] “OWL Ontology Converter.”

[32] F. Bergmann and J. Quantz, “Parallelizing description logics,” *KI-95: Advances in Artificial Intelligence*, 1995, pp. 137-148.

[33] M. Aslani and V. Haarslev, “Towards Parallel Classification of TBoxes,” *Proc. of the 2008 International Workshop on Description Logics (DL 2008*.