University of Crete

Computer Science Department

On Computing Deltas of RDF/S Knowledge Bases

Dimitris Zeginis

Master's Thesis

Heraklion, March 2008

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΡΗΤΗΣ
ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΚΑΙ ΤΕΧΝΟΛΟΓΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΕΠΙΣΤΗΜΗΣ ΥΠΟΛΟΓΙΣΤΩΝ

Υπολογισμός Διαφορών μεταξύ RDF Βάσεων Γνώσης

Εργασία που υποβλήθηκε απο τον

Δημήτρη Κ. Ζεγκίνη

ως μερική εκπλήρωση των απαιτήσεων για την απόκτηση

ΜΕΤΑΠΤΥΧΙΑΚΟΥ ΔΙΠΛΩΜΑΤΟΣ ΕΙΔΙΚΕΥΣΗΣ

Συγγραφέας:

Δημήτρης Ζεγκίνης, Τμήμα Επιστήμης Υπολογιστών

Εισηγητική Επιτροπή:

Βασίλης Χριστοφίδης, Αναπληρωτής καθηγητής, Επόπτης

Ιωάννης Τζίτζικας, Επίκουρος καθηγητής, Μέλος

Γρηγόρης Αντωνίου, καθηγητής, Μέλος

Δεκτή:

Πάνος Τραχανιάς, Καθηγητής
Πρόεδρος Επιτροπής Μεταπτυχιακών Σπουδών

Ηράκλειο, Μάρτιος 2008

On Computing Deltas of RDF/S Knowledge Bases

Dimitris Zeginis

Master's Thesis

Computer Science Department, University of Crete

Abstract

The Semantic Web (SW) is an evolving extension of the World Wide Web in which Web content can be expressed not only in natural language, but also in a format that can be read and used by software agents, thus permitting them to find, share and integrate information more easily. In order to cope with the evolving nature of the Semantic Web we need effective and efficient support for building advanced SW synchronization and versioning services. RDF Deltas, reporting the differences that exist between two RDF knowledge bases (KBs) have been proven to be crucial to reducing the amount of data that need to be exchanged and managed over the network in order to synchronize distributed Data Bases and to reduce the storage space needed for managing different versions of RDF KBs.

This Thesis analyzes the various ways in which two RDF KBs can be compared and how the result of this comparison can be used in order to transform one KB to a KB semantically equivalent to another and vice versa. The complexity of comparing RDF KBs stems from the fact that RDF graphs, in contrast to text, do not have a unique serialization and RDF KBs are also enriched with the semantic of RDFS (i.e. inferred triples). For this reason, new comparison functions are introduced and analyzed in association with the semantics of the change operations. Specifically, this Thesis identifies the pairs of (comparison function, change operation semantics) that satisfy correctness when used for synchronizing Knowledge Bases. Moreover, other desired properties of those pairs are investigated: small size of comparison result, semantic identity, non-redundancy, invertibility and composability. Finally, the algorithms for all the comparison functions are sketched and their complexity is examined. The complexity results are verified with experimental results.

In summary, the contribution of this work lies in formally defining the the comparison function and the change operation semantics, as well as the interplay between them and the investigation of the properties the pairs of comparison function and change operation semantics have.

Supervisor: Vassilis Christophides
Associate Professor
Co-Supervisor: Yannis Tzitzikas
Assistant Professor

Υπολογισμός Διαφορών μεταξύ RDF Βάσεων Γνώσης

Δημήτρης Ζεγκίνης

Μεταπτυχιακή Εργασία

Τμήμα Επιστήμης Υπολογιστών, Πανεπιστήμιο Κρήτης

Περίληψη

Ο Σημασιολογικός Ιστός είναι μια εξελισσόμενη επέκταση του Παγκόσμιου Ιστού στην οποία το περιεχόμενο μπορεί να εκφραστεί όχι μόνο με φυσική γλώσσα αλλά και με τυπικές γλώσσες που επιτρέπουν την παροχή προηγμένων υπηρεσιών αναζήτησης, διαμοιρασμού και ολοκλήρωσης πληροφορίας. Για την επιτυχή αξιοποίηση του συνεχώς αναπτυσσόμενου Σημασιολογικού Ιστού χρειάζονται αποδοτικές υπηρεσίες συγχρονισμού και διαχείρισης πολλαπλών εκδόσεων. Τα Δέλτας, που αναφέρουν τις διαφορές που υπάρχουν μεταξύ δύο σημασιολογικών βάσεων γνώσης, έχουν αποδειχτεί ότι είναι πολύ σημαντικά για να μειωθεί το μέγεθος των δεδομένων που μεταφέρονται μέσω δικτύου ώστε να συγχρονιστούν κατανεμημένες βάσεις δεδομένων και να μειωθεί ο απαιτούμενος αποθηκευτικός χώρος για την διατήρηση διαφορετικών εκδόσεων RDF βάσεων γνώσης.

Στην συγκεκριμένη εργασία αναλύονται οι διαφορετικοί τρόποι με τους οποίους μπορούν να συγκριθούν σημασιολογικές βάσεις γνώσης και πως το αποτέλεσμα αυτής της σύγκρισης μπορεί να αξιοποιηθεί για τον μετασχηματισμό μίας βάσης γνώσης σε μια βάση γνώσης σημασιολογικά ισοδύναμη με την άλλη, και το αντίστροφο. Η δυσκολία στην σύγκριση RDF βάσεων γνώσης έγκειται στο γεγονός ότι οι RDF γράφοι, σε αντίθεση με το κείμενο, δεν έχουν μοναδική σειριοποίηση και στο ότι οι RDF βάσεις γνώσης είναι εμπλουτισμένες με την σημασιολογία της RDF/S (με επαγόμενες τριπλέτες). Για τον λόγο αυτό εισάγονται και αναλύονται νέοι τελεστές σύγκρισης σε συνάρτηση με την σημασιολογία των πράξεων ενημέρωσης. Συγκεκριμένα, στην εργασία εντοπίζονται εκείνα τα ζεύγη (τελεστής σύγκρισης & σημασιολογία ενημέρωσης) που εγγυώνται ορθότητα αν εφαρμοστούν για το συγχρονισμό βάσεων γνώσης και συνάμα μελετώνται άλλες επιθυμητές ιδιότητες: μικρό μέγεθος αποτελεσμάτων σύγκρισης, ικανότητα σημασιολογικής ταυτοποίησης απουσία πλεονασμών, αντιστρεψιμότητα και σύνθεση. Τέλος, παρουσιάζονται οι αλγόριθμοι για όλους τους τελεστές

σύγκρισης και αναλύεται η πολυπλοκότητά τους. Τα αποτελέσματα της πολυπλοκότητας ενισχύονται και με πειραματικές μετρήσεις.

Συνοψίζοντας, η συνεισφορά της εργασίας έγκειται στον τυπικό προσδιορισμό των τελεστών σύγκρισης και των σημασιολογιών των πράξεων ενημέρωσης, καθώς επίσης και στην αλληλεπίδραση μεταξύ τους και στον προσδιορισμό των ιδιοτήτων που έχουν τα ζευγάρια τελεστή σύγκρισης - σημασιολογίας αλλαγής.

Επόπτης Καθηγητής: Βασίλης Χριστοφίδης
Αναπληρωτής Καθηγητής
Συνεπιβλέπον Καθηγητής: Γιάννης Τζίτζικας
Επίκουρος Καθηγητής

Ευχαριστίες

Στο σημείο αυτό θα ήθελα να ευχαριστήσω τον επόπτη καθηγητή μου κ. Βασίλη Χριστοφίδη για την άψογη συνεργασία μας τα τελευταία 2 χρόνια, καθώς επίσης και για την ουσιαστική του καθοδήγηση και συμβολή στην ολοκλήρωση της παρούσας εργασίας.

Θα ήταν μεγάλη μου παράλειψη αν παράβλεπα να εκφράσω τις ευχαριστίες μου, στον κύριο Γιάννη Τζίτζικα για τις πολλαπλές και εύστοχες παρατηρήσεις του, καθώς επίσης και για την ουσιαστική συμβολή του για την περάτωση αυτής της εργασίας. Επιπλέον, τον ευχαριστώ για τον μεθοδικό και συστηματικό τρόπο που μου έμαθε να εργάζομαι.

Επίσης θα ήθελα να εκφράσω τις ευχαριστίες μου στον καθηγητή κ. Γρηγόρη Αντωνίου για την μεγάλη προθυμία του να συμμετάσχει στην τριμελή εξεταστική επιτροπή της μεταπτυχιακής μου εργασίας.

Παράλληλα, θα ήθελα να ευχαριστώ τον Δημήτρη Χατζηνικολάου για την βοήθειά του στην διόρθωση και βελτίωση του κειμένου, καθώς επίσης και τον Γιάννη τον Θεοχάρη για την παραγωγή των συνθετικών σχημάτων που χρησιμοποιήθηκαν για τα πειράματα.

Ευχαριστίες οφείλω και στο Πανεπιστήμιο Κρήτης και το Ινστιτούτο Πληροφορικής του Ιδρύματος Τεχνολογίας και Έρευνας για τις γνώσεις και τις εμπειρίες που μου προσέφεραν όλα αυτά τα χρόνια.

Τέλος θα ήθελα να ευχαριστήσω του γονείς μου Κωνσταντίνο και Ελένη και τον αδερφό μου, Χρυσόστομο, για την υποστήριξη και την αγάπη με την οποία με περιέβαλαν.

Contents

Table of Contents iii

List of Figures viii

1 Introduction 1

1.1 Introduction to the Semantic Web 1

1.1.1 The Semantic Web Tower 1

1.1.1.1 The Representation Layer 2

1.1.1.2 RDF and Ontology Languages 3

1.2 Versioning and Synchronization Services 4

1.2.1 Versioning Services 4

1.2.2 Synchronization Services 5

1.3 Motivation for comparing of RDF KBs 7

1.3.1 Motivating Scenarios 10

1.4 Contributions 12

1.5 Organization of the thesis 12

2 Computing and Executing Deltas Between RDF KBs 13

2.1 Introduction 13

2.1.1 Preliminaries: RDF/S KBs 14

2.2 RDF/S KBs Deltas 15

2.2.1 RDF/S Delta Size 17

2.3 RDF/S KB Change Operations Semantics 20

2.3.1 Semantics of Basic Change Operations 20

2.4 Differential Functions and Change Operation Semantics 23
2.4.1 Correctness, Semantic Identify, Non-Redundancy, Reversibility and Composability of RDF Deltas 24
2.4.2 Correctness of $(\Delta_x, \mathcal{U}_y)$-pairs 25
2.4.3 Semantic Identify and Non-Redundancy of $(\Delta_x, \mathcal{U}_y)$-pairs 27
2.4.4 Reversibility of Δ_x 28
2.4.4.1 Reversibility and Delta size 29
2.4.5 Composability of Δ_x 29
2.4.6 Streaming Application/Execution of Deltas 31
2.4.7 Summarizing the Results 33

3 Algorithm Specification and Implementation 35
3.1 Introduction 35
3.2 Notations and Definitions 35
3.3 Algorithm 38
3.3.1 Mapping Algorithm 38
3.3.2 Diff Algorithm 38
3.3.3 RDF/S Delta Complexity 45
3.4 Main Memory Implementation 46
3.5 Comparison Service 48
3.6 Experimental Evaluation 51
3.6.1 Testbed 52
3.6.2 Experimental Results 53

4 Related Work 59
4.1 Introduction 59
4.2 Ontoview 60
4.2.1 Rules for Changes 62
4.3 Promptdiff 63
4.3.1 Heuristic Matchers 65
4.4 Semversion 66

4.4.1 Set-based Diff . . . 67
4.4.2 Structural Diff . . . 67
4.4.3 Semantic Diff . . . 67
4.5 RDF_utils . . . 68
4.6 CWM of w3c . . . 70
4.6.1 Patch file format . . . 71
4.7 Jena . . . 71
4.8 pOWL . . . 72
4.9 Summary Comparison . . . 73

5 Conclusion and Future Work **77**

6 Appendix: Proofs **87**
6.1 Delta Size Proofs . . . 87
6.2 Auxiliary Proofs . . . 91
6.3 Correctness Proofs . . . 91
6.4 Semantic Identity, Reversibility and Composability Proofs . . . 94
6.5 Proofs for streaming execution of deltas . . . 97

List of Tables

2.1 Two alternative change operation semantics: $\mathcal{U}_p$ and $\mathcal{U}_{ir}$ 21
2.2 Examples 26
2.3 Synopsis 33

3.1 Time complexity of the differential functions 46
3.2 Delta time for Biological Data 54
3.3 Delta size for Biological Data 55
3.4 Delta time for simple Synthetic Data 55
3.5 Delta size for simple Synthetic Data 56
3.6 Delta time for Complex Synthetic Data 56
3.7 Delta size for the Complex Synthetic Data 57

4.1 Features of the comparison 74
4.2 Features comparison of versioning systems and tools 74

List of Figures

1.1 The Semantic Web Tower 2

1.2 Transforming K to K' and vice versa 9

2.1 What set of change operations could transform K to K'? 13

2.2 Distinctions of triples sets 15

2.3 KB without unique reduction 15

2.4 Added and deleted triples of the differential functions 16

2.5 Ordering of Delta Sizes. 19

2.6 Two equivalent KBs 27

2.7 Composability of Δ_x 30

3.1 The UML diagram of the main memory implementation 47

3.2 An architectural overview of the SWKM services 49

3.3 The Comparison service sequence diagram 50

3.4 Two dependent namespaces 51

3.5 Delta time and size for simple Synthetic Data 55

3.6 Delta time and size for Complex Synthetic Data 57

3.7 (a) Size of Forward Deltas (b) Size of Forward/Backward Deltas 58

4.1 Comparing two ontologies in OntoView 61

4.2 The architecture of the PromptDiff 63

4.3 The structural diff showing the difference between two versions 64

4.4 The Layered Architecture of SemVersion 66

4.5 The granularity of the Semantic Web ranges from the universal graph to triple. 68
4.6 Three RDF graphs that show personal information from three sources. The first one asserts that a person who has first name 'Li' and surname 'Ding'. 69
4.7 pOWL Versioning. 73

Chapter 1

Introduction

1.1 Introduction to the Semantic Web

The Semantic Web is an evolving extension of the World Wide Web in which web content can be expressed not only in natural language, but also in a format that can be read and used by software agents, thus permitting them to find, share and integrate information more easily. It derives from W3C director Sir Tim Berners-Lee's vision of the Web as a universal medium for data, information, and knowledge exchange.

A set of new languages organized in a layered architecture will allow users and applications to write and share information in a machine-readable way, and will enable the development of a new generation of technologies and toolkits. This layered architecture of the Semantic Web is often referred to as the Semantic Web tower.

1.1.1 The Semantic Web Tower

The Semantic Web tower (Figure 1.1) is a work in progress, where the layers are developed in a bottom-up manner. The so far defined languages in the bottom-up order include: XML, RDF, RDF Schema and Web Ontology Language OWL. The next step in the development of the Semantic Web will be the logic and proof layer. In the next sections we will briefly describe the basic layers of the tower.

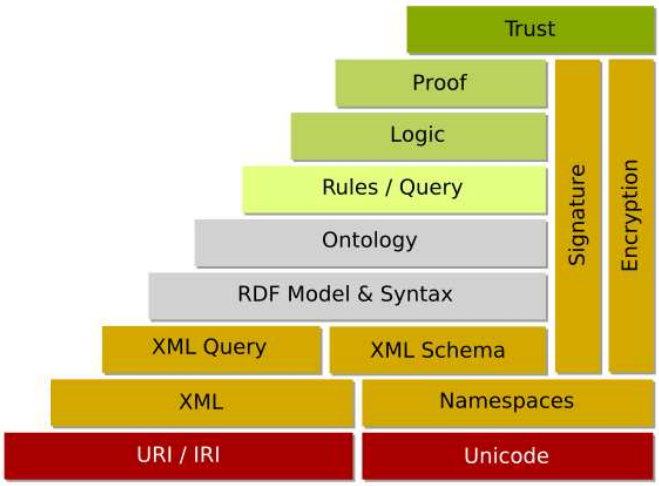


Figure 1.1: The Semantic Web Tower

1.1.1.1 The Representation Layer

The base language of the Semantic Web tower is XML, the dominant standard for data encoding and exchange on the web. In the context of the Semantic Web XML will be used for encoding any kind of data, including the meta-data, describing the meaning of application data. Such meta-data will be described by the languages of the next layers of the Semantic Web tower.

Several mechanisms have been proposed for defining sets of XML documents. A standard one is the XML Schema language. Thus, an XML schema is an XML document defining a (usually infinite) set of XML documents. This makes possible the automatic validation of a given XML document d with respect to a given schema s, that is automatic check, whether or not d is in the set of documents defined by s.

The syntax of the languages of the next layers of the Semantic Web is also defined in XML. This means that the constructs of these languages are encoded as XML documents, and can be validated against the language definitions by standard validators. However, alternative syntaxes, better suitable for the human, can be provided and can be used as a starting point for defining the semantics of these languages.

The XML Namespaces [21] and Uniform Resource Identifiers [22] are important standards used in XML and therefore also in the upper layers of the Semantic Web, which are

encoded in XML. They make it possible to create unique names for web resources.

1.1.1.2 RDF and Ontology Languages

The idea of the Semantic Web is to describe the meaning of web data in a way suitable for automatic reasoning. This means that a descriptive data (meta-data) in machine readable form is to be stored on the web and used for reasoning. The simplest form of such description would assert relations between web resources. A more advanced description, called ontology, to be shared by various applications, would define concepts in the domain of these applications. Usually an ontology defines a hierarchy of classes of objects in the described domain and binary relations, called properties.

The Semantic Web tower introduces language layers for describing resources and for providing ontologies:

- The Resource Description Framework (RDF) makes it possible to assert binary relations between resources (identified by URI's), and between resources and literals. Such assertions have the form of triples, called statements. The elements of a triple are called subject, predicate and object. However, the vocabulary of RDF does not distinguish predicate symbols from logical constants: the predicates of RDF sentences may also appear as subjects and objects. In addition, RDF allows reification of a statement which can then, for example, be used as the subject of another statement.

- For describing hierarchies of concepts RDF is extended with some built-in properties interpreted in a special way. The extension is called RDF Schema. Statements of RDF Schema (RDFS) make it possible to define hierarchies of classes, hierarchies of properties and to describe domains and ranges of the properties.

- The Web Ontology Language (OWL) builds-up on RDFS introducing more expressive description constructs. OWL has three increasingly expressive sublanguages: OWL Lite, OWL DL and OWL Full. OWL Lite supports those users primarily needing a classification hierarchy and simple constraints. OWL DL supports users who want the maximum expressiveness while retaining computational completeness

and decidability. OWL Full is meant for users who want very high expressiveness and the syntactic freedom of RDF with no computational guarantees.

1.2 Versioning and Synchronization Services

As already mentioned, Semantic Web is an evolving extension of the World Wide Web. This means that a lot of data need to be stored or exchanged over the network. Two kinds of services are exploited to cope with those needs:

- Versioning services are used to control multiple versions of the same unit of information.
- Synchronization services are used to keep remote data consistent (i.e. both of them contain the same information although something changed at one side).

1.2.1 Versioning Services

Version control is the management of multiple revisions of the same unit of information. It is most commonly used in engineering and software development to manage ongoing development of digital documents like application source code and other critical information that may be worked on by a team of people. Changes to these documents are identified by incrementing an associated number or letter code, termed the "revision number", "revision level", or simply "revision" and associated historically with the person making the change. A simple form of revision control, for example, has the initial issue of a drawing assigned the revision number "1". When the first change is made, the revision number is incremented to "2" and so on.

In computer software engineering, revision control is any practice that tracks and provides control over changes to source code. Software developers sometimes use revision control software to maintain documentation and configuration files as well as source code. Most revision control software can use delta encoding, which retains only the differences between successive versions of files. This allows more efficient storage of many different versions of files.

Delta encoding is a way of storing or transmitting data in the form of differences between sequential data rather than complete files. Delta encoding is sometimes called delta compression, particularly where archival histories of changes are required.

The differences are recorded in discrete files called "deltas" or "diffs". Because changes are often small, delta encoding greatly reduces data redundancy. Collections of unique deltas are substantially more space-efficient than their non-encoded equivalents. From a logical point of view the difference between two data values is the information required to obtain one value from the other. The difference between identical values (under some equivalence) is often called 0 or the neutral element. A good delta should be minimal, or ambiguous unless one element of a pair is present.

A delta can be defined in 2 ways, symmetric delta and directed delta. A symmetric delta can be expressed as: $\Delta(v_1, v_2) = (v_1\ v_2) \cup (v_2\ v_1)$, where v_1 and v_2 represent two successive versions.

A directed delta, also called a change, is a sequence of (elementary) change operations which, when applied to one version v_1, yield another version v_2 (note the correspondence to transaction logs in databases).

In delta encoded transmission over a network, where only a single copy of the file is available at each end of the communication channel, special error control codes are used to detect which parts of the file have changed since its previous version.

The nature of the data to be encoded influences the effectiveness of a particular compression algorithm. Delta encoding performs best when data has small or constant variation.

1.2.2 Synchronization Services

In computer science, synchronization refers to one of two distinct, but related concepts:

- Process synchronization refers to the idea that multiple processes are to join up or handshake at a certain point, so as to reach an agreement or commit to a certain sequence of action.

- Data synchronization refers to the idea of keeping multiple copies of a dataset in

coherence with one another, or to maintain data integrity. Process synchronization primitives are commonly used to implement data synchronization.

At this thesis we study the problem of data synchronization. Data synchronization is the process of establishing consistency among data on remote sources and the continuous harmonization of the data over time. It is fundamental to a wide variety of applications, including file synchronization [39], Personal Digital Assistant synchronization [1], and Public Key Server synchronization.

Several theoretical models of data synchronization exist in the research literature. The models are classified based on how they consider the data to be synchronized. Some models consider the data to be unordered while others consider the data to be ordered.

The problem of synchronizing **unordered** data (also known as the set reconciliation problem) is modeled as an attempt to compute the symmetric difference $S_A \oplus S_B = (S_A - S_B) \cup (S_B - S_A)$ between two remote sets S_A and S_B [30]. Some solutions to this problem are typified by:

- *Wholesale transfer*. In this case all data is transferred to one host for a local comparison.
- *Timestamp synchronization*. In this case all changes to the data are marked with timestamps. Synchronization proceeds by transferring all data with a timestamp later than the previous synchronization.
- *Mathematical synchronization*. In this case data are treated as mathematical objects and synchronization corresponds to a mathematical process.

Considering **ordered** data two remote strings σ_A and σ_B need to be reconcilied. Typically, it is assumed that these strings differ by up to a fixed number of edits (i.e. character insertions, deletions, or modifications). Some solution approaches to this problem include:

- shingling - splitting the strings into shingles in order to reduce this problem into an unordered synchronization problem.[8]
- synchronizing files and directories from one location to another while minimizing data transfer using delta encoding.

1.3 Motivation for comparing of RDF KBs

In order to cope with the evolving nature of the Semantic Web (SW) we need effective and efficient support for building advanced SW synchronization and versioning services. RDF Deltas, reporting the differences that exist between two RDF knowledge bases have been proven to be crucial in order to reduce the amount of data that need to be exchanged and managed over the network in this respect [23, 24, 4, 11].

Although RDF knowledge bases can be serialized in various text formats (e.g., XML[1], N-Triples[2], Trix[3]), a straightforward application of existing version control systems for software code, such as RCS [38] and CVS [6], or for XML data, such as [29], [12] and [9], is not a viable solution for computing RDF Deltas. This is mainly due to the fact that RDF KBs, essentially represent graphs which (a) may feature several possible serializations (since there is no notion of edge ordering in [6]) and (b) are enriched with the semantics of RDFS specification (also including inferred edges according to [7]). For these reasons, several non text-based tools have been recently developed for comparing RDF graphs produced autonomously on the SW, as for example, SemVersion [41], PromptDiff [32], Ontoview [24], [15] and [4]. In most cases, the output of these tools is exploited by humans, and thus an intuitive presentation of the comparison results (and other related issues) has received considerable attention. SemVersion [41] proposes two Diff algorithms: (a) one *structure-based* which returns a set-based difference of the triples explicitly forming the two graphs, and (b) one *semantic-aware* which also takes into account the triples inferred by the associated RDFS schemas. PromptDiff [32, 33, 31] is an ontology-versioning environment, that includes a version-differential algorithm (based on heuristic matchers [32, 33]), while the visualization of the computed difference between two ontologies is discussed in [31]. Ontoview [24] is an ontology management system, able to compare two ontology versions and highlight their differences. Notably, it allows users to specify the conceptual relations (i.e. equivalence, subsumption) between the different versions of an ontology concept. Moreover, [15, 19] introduce the notion of RDF molecules as the finest

[1] http://www.w3.org/TR/rdf-syntax-grammar/
[2] http://www.w3.org/2001/sw/RDFCore/ntriples/
[3] http://www.w3.org/2004/03/trix/

components to be used when comparing RDF graphs (in the absence of blank nodes each triple constitutes a molecule). Finally, tracking the evolution of ontologies when changes are preformed in more controlled environments (e.g. collaborative authoring tools) has been addressed in [25, 34, 42].

However, existing RDF differential tools have not yet focused on the size of the produced Deltas, a very important aspect for building versioning services over SW repositories [36]. In this thesis we are interested in computing RDF Deltas as sets of change operations (i.e. SW update programs) that enable us to transform one RDF KB into the other. Consider, for example, the two RDF KBs K and K' of Figure 1.2 and their standard representation as sets of explicitly defined triples [7]: what set of change operations could transform K to K' ($\Delta(K \rightarrow K')$) or vice versa ($\Delta(K' \rightarrow K)$)?

To answer this question we need to consider the semantics of the update primitives such as $Add(t)$ and $Del(t)$ where t is triple involving any RDF predicate. By assuming a side-effect free semantics for these primitives, i.e. $Add(t)$ (resp. $Del(t)$) is a straightforward addition (resp. deletion) of t to the set $Triples(K)$, K' can be obtained by executing the following set Δ_e (e stands for $e$xplicit) of change operations:

$$\begin{aligned}\Delta_e \;=\; \{&Del(TA\ subClassOf\ Person),\ Del(TA\ subClassOf\ Student),\\ &Del(Jim\ \ type\ \ TA),\ Add(TA\ subClassOf\ Graduate),\\ &Add(Student\ subClassOf\ Person),\ Add(Jim\ \ type\ \ Person)\}\end{aligned}$$

Δ_e is actually composed of update operations over the explicit triples of K and K', and it is provided by the majority of existing RDF differential tools [4, 41, 15]. However, by assuming side-effects (on the inferred triples not represented in Figure 1.2) during the execution of the above update primitives, we can reach K' by applying on K the following set Δ_d (d stands for $d$ense) of change operations:

$$\begin{aligned}\Delta_d \;=\; \{&Del(Jim\ \ type\ \ TA),\ Add(TA\ subClassOf\ Graduate),\\ &Add(Student\ subClassOf\ Person)\}\end{aligned}$$

As we can easily observe, Δ_d has only three change operations in contrast to Δ_e that has six, given that inferred triples are also taken into account for the Delta computation. For example, $Del(TA\ subClassOf\ Person)$ is not included in Δ_d because it can be inferred

from K'. As we can see in Figure 1.2, this differential function yields even smaller in size operation sets than the Δ_c (c stands for $c$losure) semantics-aware Delta of [41]. However, Δ_d cannot always successfully transform one RDF KB to another. Returning to our example of Figure 1.2, Δ_d cannot be used to migrate backwards from K' to K since $Del(Graduate\ subClassOf\ Person)$ is an operation not included in Δ_d. For this reason, we need to consider additional RDF differential functions involving inferred triples such as Δ_{dc} (dc stands for $d$ense & $c$losure) illustrated in Figure 1.2. Still the resulting sets of operations have at most the same size as those returned by Δ_c. Finally, we consider one more RDF differential function named Δ_{ed} (ed stands for $e$xplicit & $d$ense) illustrated in Figure 1.2. Still the resulting sets of operations have at most the same size as those returned by Δ_e.

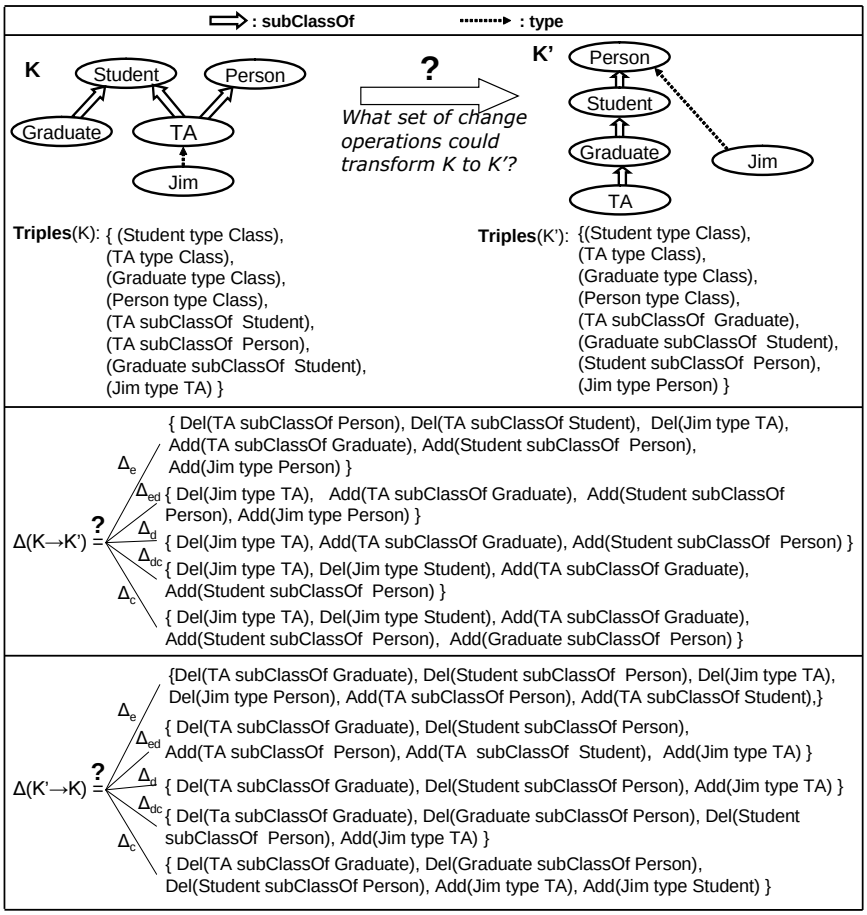


Figure 1.2: Transforming K to K' and vice versa

RDF differential functions that yield as less as possible change operations are quite

beneficial to building SW versioning services. In particular, by advocating a change-based versioning framework [13] we can store in a SW repository only the update programs required to migrate (forward or backward) from one version to another rather than the entire set of triples for each version. In a nutshell, storing (or exchanging) as less as possible change operations is more space- (or time-) efficient. In this context, the main questions addressed by our work are: (a) what semantics of update primitives would make the above scenario possible (i.e. with what side-effects), and (b) how could we compute the corresponding set of change operations (i.e. with what differential functions)?

1.3.1 Motivating Scenarios

One of the primary applications of RDF/S Deltas is for efficiently storing multiple versions of a KB in a CVS[4]-like archive. Rather than storing complete snapshots of all KB versions, deltas between consecutive KB versions are usually organized in a chain. If a delta's target version is newer than its source version in the archive, then it is refereed as *forward delta*. A *backward delta* has the opposite orientation. In this context, several delta-based version archiving policies have been proposed in the literature [14]:

- *Age dependent forward strategy:* only the oldest version of a KB (i.e. K_0) is stored, and all the other versions are accessed using forward deltas. The deltas are computed between consecutive versions i.e. $\Delta(K_0 \rightarrow K_1)$, $\Delta(K_1 \rightarrow K_2)$, ..., $\Delta(K_{n-1} \rightarrow K_n)$. Obviously, the older a version is the faster will the access be. To access the latest version, we have to materialize it in $n-1$ steps: $\Delta(K_0 \rightarrow K_1)$, $\Delta(K_1 \rightarrow K_2)$, ..., $\Delta(K_{n-1} \rightarrow K_n)$.

- *One step forward strategy:* only the oldest version of a KB (i.e. K_0) is stored while deltas are computed between the oldest and every other forward version i.e. $\Delta(K_0 \rightarrow K_1)$, $\Delta(K_0 \rightarrow K_2)$, ..., $\Delta(K_0 \rightarrow K_n)$. In this way every version, and thus the latest one too, can be accessed in one step. Access time to a version therefore is not dependent on its age. Obviously, the size of the deltas computed with this

[4] www.cvshome.org

strategy is bigger that the size of the deltas computed with the *age dependent forward strategy* (e.g. $|\Delta(K_{n-1} \rightarrow K_n)| < |\Delta(K_0 \rightarrow K_n)|$).

- *Age dependent backward strategy:* only the latest version of a KB (i.e. K_n) is stored and all the other versions are accessed using backward deltas. The deltas are computed between consecutive versions i.e. $\Delta(K_n \rightarrow K_{n-1})$, $\Delta(K_{n-1} \rightarrow K_{n-2})$, ..., $\Delta(K_1 \rightarrow K_0)$. Access to the latest version is supported without any additional penalty. Access time to versions will grow with their age, which seems to be a reasonable assumption.

- *One step backward strategy:* only the latest version of a KB (i.e. K_n) is also stored while deltas are computed between the latest version and every other backward version i.e. $\Delta(K_n \rightarrow K_{n-1})$, $\Delta(K_n \rightarrow K_{n-2})$, ..., $\Delta(K_n \rightarrow K_0)$. In this way every version, and thus the oldest one too, can be accessed in one step. This strategy seems to be the ideal choice at first sight; however, all the deltas have to be recomputed completely whenever a new version is created. This stems from the fact that deltas are related to the latest version which changes whenever a new version is created.

Clearly, the storage requirement of the employed RDF/S Deltas between consecutive KB versions is crucial. *Small sized* Deltas yielding as less as possible change operations are quite beneficial to all above versioning policies. In extremis, RDF/S Deltas should not report any change between two *semantically equivalent* RDF/S KBs (i.e. with the same set of explicit and inferred triples). Additional requirements arise when we want to propagate changes across distributed RDF/S KB versions (i.e. synchronization). In this setting, a CVS-like archive should be able to aggregate RDF/S deltas and apply them to a KB for which the deltas involved were not computed. For example, if changes for a version K_1 should be propagated to a K_2 in another archive, we need to determine the nearest common ancestor K of versions K_1 and K_2. Then, the backward delta sequence leading from K to K_1 should be aggregated with the forward delta sequence leading from K to K_1 and the resulting delta should be applied to K (regardless of the versioning policy employed by the archives). In this respect, we also need RDF/S Deltas that can be *reversed* and *composed* without requiring to materialize the involved KB versions. As

we will see in the rest of the thesis these requirement cannot be satisfied by minimum in size RDF/S Deltas because they incur loss of information.

1.4 Contributions

In a nutshell the main contributions of this thesis are:

- We analyze five different differential functions returning sets of changes operations, namely, *explicit* (Δ_e), *closure* (Δ_c), *dense* (Δ_d), *dense & closure* (Δ_{dc}), and *explicit & dense* (Δ_{ed}) .
- We introduce two change operations semantics: one *plain* set-theoretic ($\mathcal{U}_p$), and another that involves *inference* and *redundancy elimination* ($\mathcal{U}_{ir}$).
- We study which combinations of change operation semantics and differential functions are correct and satisfy properties such as semantic identity non redundancy, reversibility and composability.
- The algorithms that implement the five comparison functions are sketched
- Interesting experimental results are reported

1.5 Organization of the thesis

Chapter 2 introduces five differential functions and two change operation semantics and analyzes their complexity. Then the pairs comparison function - change operation semantics are checked for some desired properties.

Chapter 3 presents the algorithms used for each of the five comparison functions. Furthermore, some details assuming the main memory and web service implementation are also provided.

Chapter 4 examines the state of the art in tools used to compare Knowledge Bases in the Semantic Web.

Chapter 5 summarizes the results of this thesis and identifies topics that are worth further work and research.

Chapter 2

Computing and Executing Deltas Between RDF KBs

2.1 Introduction

Versioning and Synchronization services need a method to compare two RDF KBs and then transform the first KB to the other (Figure 2.1). For this reason two modules are required:

- A differential function to report the differences between two RDF KBs
- A change operation semantics that indicates the way the differences must be applied to the first RDF KB to get the second one.

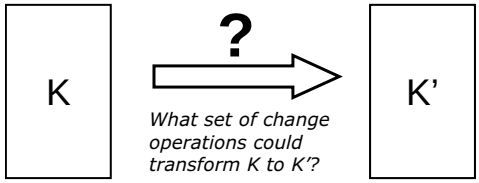


Figure 2.1: What set of change operations could transform K to K'?

Obviously, a differential function that yields the smallest in size result is preferred. Further, more a pair of (differential function & change operation semantics) must at least satisfy correctness when synchronizing remote Knowledge Bases.

2.1.1 Preliminaries: RDF/S KBs

In general, an RDF/S Knowledge Base (KB) is defined by a set of triples of the form (subject, predicate, object). Let $\mathcal{T}$ be the set of all possible triples that can be constructed from an infinite set of URIs (for resources, classes and properties) as well as literals [21]. Then, a KB can be seen as a finite subset K of $\mathcal{T}$, i.e. $K \subseteq \mathcal{T}$. Apart from the explicitly specified triples of a K, other triples can be inferred based on the RDF/S semantics [22]. For this reason, we introduce the notion of closure and reduction of RDF/S KBs.

The *closure* of a K, denoted by $C(K)$, contains all the triples that either are explicitly asserted or can be inferred from K by taking into account class or property assertions made by the associated RDFS schemas. Thus, we can consider that $C(K)$ is defined (and computed) by taking the reflexive and transitive closures of RDFS binary relations such as subClassof and type. It should be stressed that our work is orthogonal to the consequence operator of logic theories [17] actually employed to define the closure operator C. Specifically, if P denotes the powerset of all possible sets of triples of $\mathcal{T}$, then the closure operator can be defined as any function $C : P \rightarrow P$ that satisfies the following properties:

- $K \subseteq C(K)$ for all K, i.e. C is *extensive*
- If $K \subseteq K'$ then $C(K) \subseteq C(K')$, i.e. C is *monotonically increasing*
- $C(C(K)) = C(K)$ for all K, i.e. C is an *idempotent* function

If it holds $C(K) = K$, then we will call K *completed*. The elements of K will be called *explicit* triples, while the elements of $C(K) - K$ will be called *inferred*. We can now define an equivalence relation between two knowledge bases.

Def. 1 Two knowledge bases K and K' are *equivalent*, denoted by $K \sim K'$, iff $C(K) = C(K')$.

The *reduction* of a K, denoted by $R(K)$, is the smallest in size set of triples such that $C(R(K)) = C(K)$. Let Ψ denote the set of all knowledge bases that have a unique reduction. Independently of whether the reduction of a K is unique or not, we can characterize a K as (semantically) *redundancy free*, and we can write $RF(K) = True$

(or just $RF(K)$), if it does not contain explicit triples which can be inferred from K. Formally, K is redundancy free if there is not any proper subset K' of K (i.e. $K' \subset K$) such that $K \sim K'$. Figure 2.2 illustrates the above sets of triples ($R(K)$ is enclosed in a dashed box because it is not always unique).

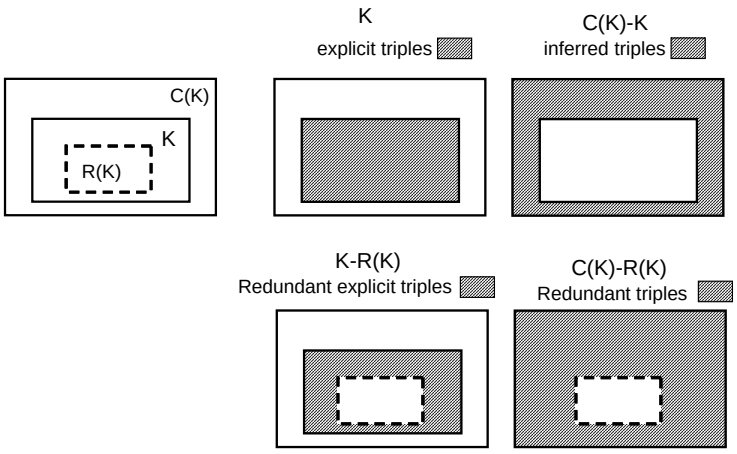


Figure 2.2: Distinctions of triples sets

It is worth noticing that the reduction of a K is not always unique. In general, uniqueness of the transitive reduction of a binary relation R is guaranteed only when R is antisymmetric and finite. Unfortunately, this is not the case of RDF/S KBs allowing cycles in the subsumption relations. For example, in Figure 2.3 we have $K \sim K_1 \sim K_2$, moreover $RF(K_1), RF(K_2)$, but $K_1 \neq K_2$.

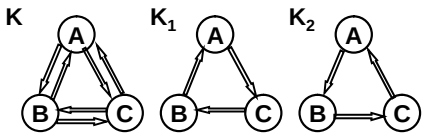


Figure 2.3: KB without unique reduction

2.2 RDF/S KBs Deltas

In this Section we formally define the five differential functions of RDF/S KBs introduced in Figure 1.2, namely, Δ_e, Δ_c, Δ_d, Δ_{dc} and Δ_{ed}.

$$
\begin{aligned}
\Delta_e(K \to K') &= \{Add(t) \mid t \in K' - K\} \quad \cup \{Del(t) \mid t \in K - K'\} \\
\Delta_c(K \to K') &= \{Add(t) \mid t \in C(K') - C(K)\} \cup \{Del(t) \mid t \in C(K) - C(K')\} \\
\Delta_d(K \to K') &= \{Add(t) \mid t \in K' - C(K)\} \quad \cup \{Del(t) \mid t \in K - C(K')\} \\
\Delta_{dc}(K \to K') &= \{Add(t) \mid t \in K' - C(K)\} \quad \cup \{Del(t) \mid t \in C(K) - C(K')\} \\
\Delta_{ed}(K \to K') &= \{Add(t) \mid t \in K' - K\} \quad \cup \{Del(t) \mid t \in K - C(K')\}
\end{aligned}
$$

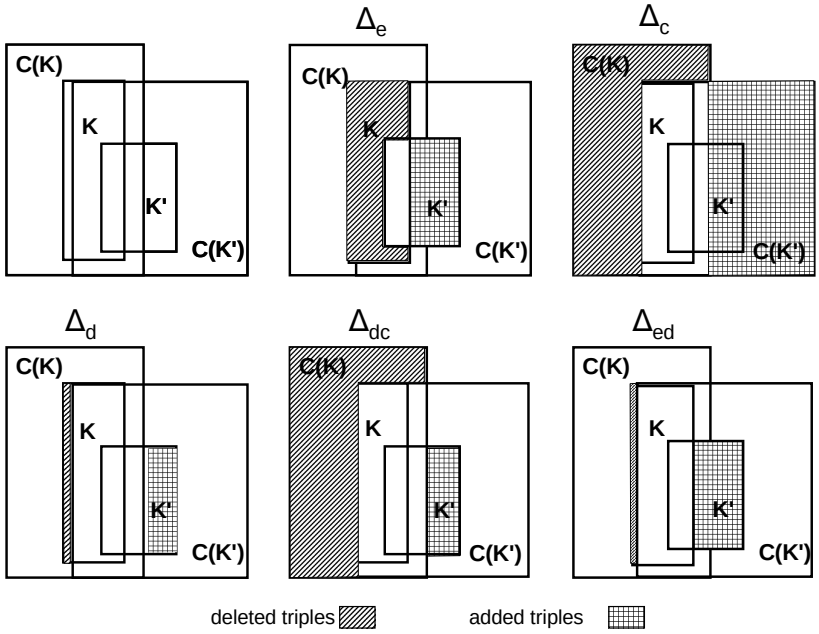


Figure 2.4: Added and deleted triples of the differential functions

Δ_e (where e stands for *e*xplicit) actually returns the set difference over the explicitly asserted triples, while Δ_c (where c stands for *c*losure) returns the set difference by also taking into account the inferred triples [1]. As we mentioned in Section 1.3, existing approaches (e.g. [41]) are based on Δ_e and Δ_c. However, as we are interested in small sized Deltas, we introduce three novel differential functions namely Δ_d (where d comes from *d*ense), Δ_{dc} (dc comes from *d*ense & *c*losure) and Δ_{ed} (ed comes from *e*xplicit & *d*ense). It is not hard to see that Δ_d produces the smallest in size set of change operations. Figure 2.4

[1] Mention that Δ_e and Δ_c define a symmetric set difference.

shows the Venn diagrams of the corresponding sets of triples to be added and deleted, in the general case of K and K'' overlapping. Unfortunately, and as we will see at Section 2.4, Δ_d can be actually applied to transform K to K'' only under certain conditions. For this reason, we additionally consider Δ_{dc} and Δ_{ed} yielding smaller in size deltas than Δ_c and Δ_e respectively. Δ_{dc} resembles Δ_d regarding additions and Δ_c regarding deletions, while Δ_{ed} resembles Δ_e regarding additions and Δ_d regarding deletions.

2.2.1 RDF/S Delta Size

Let $|\Delta(K \rightarrow K')|$ denote the *number* of change operations in $\Delta(K \rightarrow K')$. To keep notation simple we shall also use $|\Delta_x|$ to denote $|\Delta_x(K \rightarrow K')|$ for any $x \in \{e, c, d, dc, ed\}$.

Lemma 1 For any pair of knowledge bases K and K' it holds:

$$\begin{aligned} |\Delta_d| &\leq |\Delta_{ed}| \leq |\Delta_e| \\ |\Delta_d| &\leq |\Delta_{dc}| \leq |\Delta_c| \end{aligned}$$

In a nutshell, Δ_d gives always the smallest in size Deltas, Δ_{dc} gives smaller Deltas than Δ_c, Δ_{ed} gives smaller Deltas than Δ_e, while Δ_{dc} is incomparable to Δ_{ed}. Figure 2.5(a) illustrates the Hasse diagram of the ordering relation of Delta sizes in the general case.

Under specific conditions regarding K, K', $C(K)$ and $C(K')$ the above definitions may coincide.

Prop. 1 If $K \subseteq K'$ then the following relationships also hold:

$$\begin{aligned} |\Delta_e| &= |\Delta_{ed}| \\ |\Delta_d| &= |\Delta_{dc}| \end{aligned}$$

This case corresponds to the quite frequent scenario where K is stored in KB (and therefore it might be also redundancy free), and K' is derived from K by adding new triples. Hence, Δ_d and Δ_{dc} give the same in size Deltas, as also happens for Δ_e and Δ_{ed}. Figure 2.5(b) shows the corresponding Hasse diagram.

Prop. 2 If $K \supseteq K'$ then the following relationships also hold:

$$\begin{aligned} |\Delta_d| &= |\Delta_{ed}| \\ |\Delta_c| &= |\Delta_{dc}| \end{aligned}$$

This case corresponds to the scenario where K' is generated only by deleting triples from K. Hence, Δ_d and Δ_{ed} (alternately Δ_c and Δ_{dc}) give the same in size Deltas. Figure 2.5(c) shows the corresponding Hasse diagram.

Prop. 3 If $K = C(K)$ then the following relationships also hold:

$$|\Delta_d| = |\Delta_{dc}| = |\Delta_{ed}|$$

This case corresponds to the scenario where either the subsumption hierarchy of K consists of only one level (i.e. no triples can be inferred) or the closure of K is already materialized. Thus, Δ_d, Δ_{dc} and Δ_{ed} give the same in size Deltas. Figure 2.5(d) shows the corresponding Hasse diagram.

Prop. 4 If $K' = C(K')$ then the following relationships also hold:

$$\begin{aligned} |\Delta_c| &= |\Delta_{dc}| \\ |\Delta_e| &= |\Delta_{ed}| \end{aligned}$$

This case corresponds to the scenario where either the subsumption hierarchy of K' consists of only one level (i.e. no triples can be inferred) or the closure of K' is already materialized. Thus, Δ_c and Δ_{dc} give the same in size Deltas, as also happens for Δ_e and Δ_{ed}. Figure 2.5(e) shows the corresponding Hasse diagram.

Prop. 5 If $K = C(K')$ then the following relationships also hold:

$$\begin{aligned} |\Delta_d| = |\Delta_{dc}| &= |\Delta_c| = |\Delta_{ed}| = 0 \\ |\Delta_e| &\geq 0 \end{aligned}$$

In this case K is semantically equivalent to K'. This case arises when K contains explicit triples that became implicit in K'. Hence, unlike Δ_e, the differential functions Δ_c, Δ_{dc}, Δ_c and Δ_{ed} give an empty result. Figure 2.5(f) shows the corresponding Hasse diagram.

Prop. 6 If $K' = C(K)$ then the following relationships also hold:

$$\begin{aligned} |\Delta_d| = |\Delta_{dc}| &= |\Delta_c| = 0 \\ |\Delta_e| = |\Delta_{ed}| &\geq 0 \end{aligned}$$

In this case K' contains explicit triples that were implicit in K'. Thus, Δ_d, Δ_{dc} and Δ_c give an empty result, while Δ_e and Δ_{ed} give the same non-empty result. Figure 2.5(g) shows the corresponding diagram.

Prop. 7 If $K \sim K'$ then the following relationships also hold:

$$\begin{aligned} |\Delta_d| = |\Delta_{dc}| &= |\Delta_c| = 0 \\ 0 \leq |\Delta_{ed}| &\leq |\Delta_e| \end{aligned}$$

In this case K and K' are equivalent. Thus, Δ_c, Δ_{dc} and Δ_c give an empty result, while Δ_e and Δ_{ed} give a non-empty result with the Δ_{ed} producing a smaller in size result. Figure 2.5(h) shows the corresponding diagram.

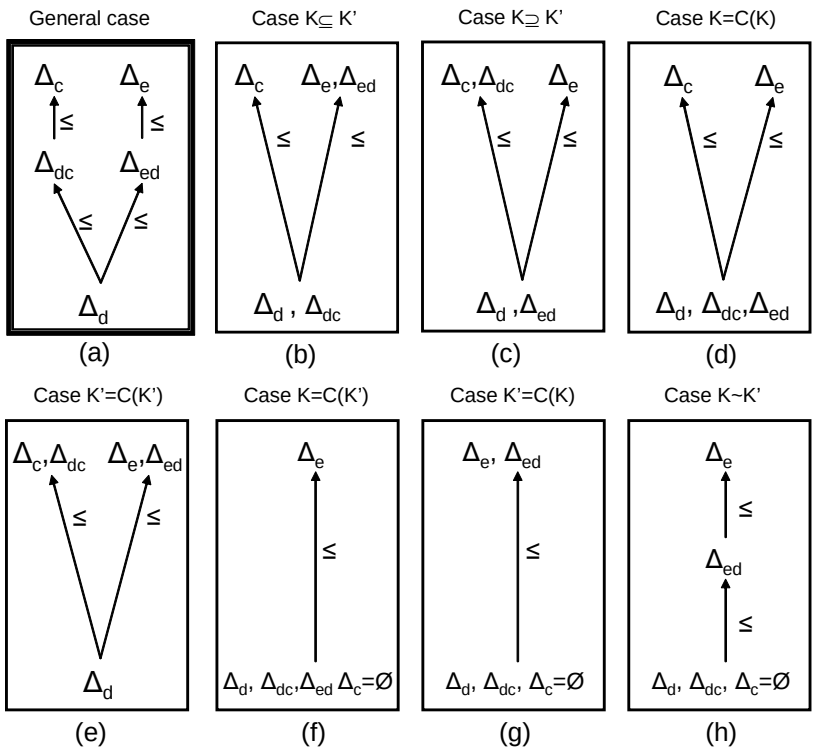


Figure 2.5: Ordering of Delta Sizes.

2.3 RDF/S KB Change Operations Semantics

In this work, we focus on two *basic* change operations allowing to transform one KB to another, namely triple *addition* $Add(t)$ and *deletion* $Del(t)$ where $t \in T$. In this respect, a triple *update* is "split" into an addition and a deletion of triplets having the same *subject* and *predicate* (and thus keep both "old" and "new" values usually ignored by updates). The five differential functions presented in the Section 2.2, yield essentially *sets of atomic change operations*. More formally, for a differential function $\Delta_x(K \rightarrow K'))$ where $x \in \{e, c, d, dc, ed\}$, Δ_x^+ is used to denote the corresponding set of triple additions (i.e. *incremental* changes) and Δ_x^- the set of triple deletions (i.e. *decremental* changes). Obviously, Δ_x contains only sets of useful change operations reflecting the *net effect* of successive modifications over the same (explicit or inferred) triple of two KB versions. In other terms, Δ_x does not contain both $Add(t)$ and $Del(t)$ operations for a given $t \in T$.

Def. 2 A Delta $\Delta(K \rightarrow K')$ is *useful* if it holds $\Delta^+ \cap \Delta^- = \emptyset$.

By defining RDF/S Deltas as sets of atomic change operations, we avoid to specify an execution order as in an edit-script (i.e. a sequence of triple additions or deletions). This design choice amends to simpler computation requirements for RDF/S Deltas while it provides the opportunity of applying alternative semantics of changes when transforming one KB to another (i.e. with or without side-effects on the KB closure).

In the sequel we will formally introduce the *semantics* of basic $Add(t)$ and $Del(t)$ operations (i.e. the exact pre- and post-conditions of each operation) while in Section 2.4.6 we will investigate how a set of change operations can be transformed to an edit-script amendable to a sequential execution.

2.3.1 Semantics of Basic Change Operations

Table 2.1 defines two alternative semantics for triple additions and deletions, namely, $\mathcal{U}_p$ (p comes from $p$lain), and $\mathcal{U}_{ir}$ (ir comes from $i$nference & $r$eduction).

- Under $\mathcal{U}_p$-semantics, change operations capture essentially plain set theoretic additions and deletions of triples. This implies that only the explicit triples are taken

into account while inferred ones are ignored (as in a standard database context).

- Under $\mathcal{U}_{ir}$-semantics, change operations incur also interesting side-effects such as redundancy elimination and knowledge preservation. This implies that the updated KB will not contain any explicit triple which can be inferred, while preserves as much of the knowledge expressed in K as possible (reminiscent of the postulates of the AGM theory [2] regarding contraction, and compliant with the semantics of the RUL update language [28]).

$\mathcal{U}_p$-semantics is straitforward. We will illustrate in the sequel $\mathcal{U}_{ir}$-semantics using the example of Figure 1.2. If we apply on K the set Δ_{dc} under $\mathcal{U}_{ir}$-semantics, then we will indeed get K'. The insertion of ($Student\ subClassOf\ \ Person$) makes the triple ($TA\ subClassOf\ Person$) redundant, so the execution of $Add(Student\ subClassOf\ Person)$ will remove ($TA\ \ subClassOf\ \ Person$) from the KB. Analogously, the insertion of ($TA\ subClassOf\ Graduate$) makes the triple ($TA\ \ subClassOf\ \ Student$) redundant, while the deletion of the triple ($Jim\ \ type\ Student$) will add the triples ($Jim\ \ type\ \ Person$) and ($Jim\ \ type\ \ Student$). Finally, the deletion of the triple ($Jim\ \ type\ \ Student$) will not have any side effects.

Change Operation Semantics $\mathcal{U}_p$				
Operation		Pre-condition	Post-condition	Comment
$Add(t)$	1	$t \in K$	$K' = K$	void
	2	$t \in C(K) - K$	$K' = K \cup \{t\}$	addition (although already inferred)
	3	$t \notin C(K)$	$K' = K \cup \{t\}$	addition
$Del(t)$	4	$t \in K$	$K' = K - \{t\}$	deletion
	5	$t \in C(K) - K$	$K' = K$	an inferred triple cannot be deleted
	6	$t \notin C(K)$	$K' = K$	void
Change Operation Semantics $\mathcal{U}_{ir}$				
$Add(t)$	7	$t \in K$	$K' = K$	void
	8	$t \in C(K) - K$	$K' = K$	it is already inferred so it is ignored
	9	$t \notin C(K)$	$K' = R(K \cup \{t\})$	addition and then reduction
$Del(t)$	10	$t \in K$	$K' = R(C(K) - \{t\})$	deletion from closure and then reduction
	11	$t \in C(K) - K$	$K' = K$	an inferred triple cannot be deleted
	12	$t \notin C(K)$	$K' = K$	void

Table 2.1: Two alternative change operation semantics: $\mathcal{U}_p$ and $\mathcal{U}_{ir}$

Returning to Table 2.1, for every incremental ($Add(t)$) or decremental ($Del(t)$) operation three different, and mutually exclusive, pre-conditions are examined, namely $t \in K$, $t \in C(K) - K$ and $t \notin C(K)$. The post-conditions of each case are specified. K (resp.

K') denotes the knowledge base before (resp. after) the execution of a change operation. Notice that post-conditions define exactly what K' will be, unless the reduction is not unique.

In particular, let t be a triple whose addition is requested. If $t \in K$, then under both $\mathcal{U}_p$ and $\mathcal{U}_{ir}$ semantics no change will be made i.e. $K' = K$ (recall that K is a *set* of triples). If $t \in C(K) - K$, then under $\mathcal{U}_p$-semantics, K' will indeed contain t however, under $\mathcal{U}_{ir}$-semantics we will have $K' = K$ because every triple that exists at $C(K) - K$ can be inferred (and $\mathcal{U}_{ir}$ aims at redundancy-free KBs). Finally, when requesting the addition of a triple $t \notin C(K)$ under $\mathcal{U}_p$, K' will contain t. Under $\mathcal{U}_{ir}$, K' will contain only the triples remaining after the elimination of redundant ones (i.e. those that can be inferred) once t is added to K.

Let's now consider the deletion of a triple t. If t belongs to K, then K' will not contain t under $\mathcal{U}_p$-semantics. Under $\mathcal{U}_{ir}$, K' will contain the triples that remain after deleting t from $C(K)$ and eliminating the redundant ones (note that $C(K)$ is used in order to preserve as much knowledge as possible). Now if $t \in C(K) - K$, then this deletion request has no effect under both semantics. This means that under both semantics, *only explicit triples can be deleted.* This relieves us from having to decide which of the (possibly several) policies need to be adopted for reaching a K' whose closure does not contain t. Finally, if $t \notin C(K)$, then this deletion request has no effect as t is already out of K.

We can now define when a KB K satisfies a basic change operations under $\mathcal{U}_p$ and $\mathcal{U}_{ir}$-semantics.

Def. 3 We will say that K *satisfies* under $\mathcal{U}_p$-semantics:
(a) an operation $Add(t)$, iff $t \in K$,
(b) an operation $Del(t)$, iff $t \notin K$

Def. 4 We will say that K *satisfies* under $\mathcal{U}_{ir}$-semantics:
(a) an operation $Add(t)$, iff $t \in C(K)$,
(b) an operation $Del(t)$, iff $t \notin C(K)$

The above definitions stem directly form the pre and post conditions of operations depicted in Table 2.1. In general, we can say that a K satisfies a RDF/S delta Δ under

$\mathcal{U}_y$, where $y \in \{p, ir\}$, iff K satisfies *every* addition of Δ^+ and deletion of Δ^-. Recall also at this point that the RDF/S deltas considered in our work consists only of useful change operations.

Let $\mathcal{S}$ be the set of all possible change operations. Let S be a finite subset of $\mathcal{S}$ (i.e. $S \subset \mathcal{S}$). If $\mathcal{U}$ is a symbol that denotes the semantics of a particular change operation (i.e. $\mathcal{U}_p, \mathcal{U}_{ir}$), then we will use $S^{\mathcal{U}}(K)$ to denote the result of applying S to K under $\mathcal{U}$ semantics. Notice that the result of applying an operation is unique under $\mathcal{U}_p$-semantics. This is true also for $\mathcal{U}_{ir}$ if we are in Ψ (KBs with unique reduction).

Def. 5 A delta $\Delta(K \rightarrow K')$ can be applied on K as follows:
(a) Under $\mathcal{U}_p$-semantics, $\Delta(K \rightarrow K')^{\mathcal{U}_p}(K) = (K - \Delta^-) \cup \Delta^+$
(b)Under $\mathcal{U}_{ir}$-semantics, $\Delta(K \rightarrow K')^{\mathcal{U}_{ir}}(K) = R((C(K) - \Delta^-) \cup \Delta^+)$

It is clear that the resulting KB always satisfies Δ_x under $\mathcal{U}_p$ and $\mathcal{U}_{ir}$-semantics for every $x \in \{e, c, d, dc, ed\}$. However, as we will see in the next Section, the resulting KB is not always a correct transformation of the original KB (see for example Table 2.2(C) for Δ_d).

2.4 Differential Functions and Change Operation Semantics

In this section we investigate which of the five differential functions (introduced in Section 2.2) and under what semantics of update primitives (presented in Section 2.3) could be used for versioning and synchronization. To this end, we formally define the notion of *correctness*, as well as other useful properties that we call *semantic identity*, *redundancy*, *reversibility* and *composability*.

Subsequently, we will elaborate on the execution of the update programs. Finally, we will identify the pairs that are correct and the properties that they satisfy.

2.4.1 Correctness, Semantic Identify, Non-Redundancy, Reversibility and Composability of RDF Deltas

Let Δ_x be a differential function, and $\mathcal{U}_y$ be a change operation semantics.

Def. 6 A pair $(\Delta_x, \mathcal{U}_y)$ is *correct* if for any pair of knowledge bases K and K', it holds $\Delta_x(K \to K')^{\mathcal{U}_y}(K) \sim K'$.

Obviously, a pair $(\Delta_x, \mathcal{U}_y)$ can be used for versioning or synchronization services only if it is correct. Apart from correctness, a pair $(\Delta_x, \mathcal{U}_y)$ may also satisfy the following properties.

(P1) **Semantic Identity**

If $K \sim K'$ then $\Delta_x(K \to K') = \emptyset$

It is desirable to have a differential function that reports an empty result if its operands are equivalent. Notice that this property is independent of the change operation semantics. It characterizes differential functions.

(P2) **Non-Redundancy**

$RF(\Delta_x(K \to K')^{\mathcal{U}_y}(K))$

It means that the execution of a delta will result in a KB that is always redundancy free (independently of whether K and K' are redundancy free or not).

(P2.1) **Weaker Non-Redundancy**

If $RF(K')$ then $RF(\Delta_x(K \to K')^{\mathcal{U}_y}(K))$

It means that if K' is RF then the resulting KB will be redundancy free too. Notice that (P2.1) is weaker than (P2): if (P2) holds then (P2.1) holds too.

(P3) **Reversibility** Let u be a change operation. We define its reversion as: $Inv(Add(t)) = Del(t)$ and $Inv(Del(t)) = Add(t)$. We generalize and define the reversion of a set of change operations U as follows: $Inv(U) = \cup\{\ Inv(u) \mid u \in U\}$. A differential function Δ_x is reversible if:

$$Inv(\Delta_x(K \to K')) = \Delta_x(K' \to K)$$

(P4) **Composability**

Let $\Delta_1 = \Delta_1^+ \cup \Delta_1^-$ and $\Delta_2 = \Delta_2^+ \cup \Delta_2^-$ be two deltas. The composition of these two deltas, denoted $\Delta_1 \circ \Delta_2$ is a delta $\Delta = \Delta^+ \cup \Delta^-$ where:

- $\Delta^+ = \Delta_1^+ \cup \Delta_2^+ - \Delta_1^- \cup \Delta_2^-$
- $\Delta^- = \Delta_1^- \cup \Delta_2^- - \Delta_1^+ \cup \Delta_2^+$

A differential function Δ_x satisfies composability if:

$$\Delta_x(K_1 \to K_2) \circ \Delta_x(K_2 \to K_3) \circ ... \circ \Delta_x(K_{n-1} \to K_n) = \Delta_x(K_1 \to K_n)$$

2.4.2 Correctness of $(\Delta_x, \mathcal{U}_y)$-pairs

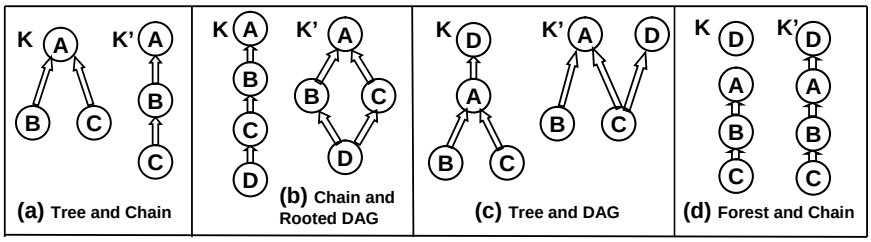
For identifying the pairs that are correct, Table 2.2 depicts 4 examples. For each example, it shows the result of applying Δ_d, Δ_c, Δ_e, Δ_{dc} and Δ_{ed} for both $K \to K'$ and $K' \to K$, and contains the following columns:

- $\mathcal{U}_p$: If Y then this means that $\Delta_x(K \to K')^{\mathcal{U}_p}(K) \sim K'$ i.e. the approach is *correct*. Otherwise the cell is marked with N. If the approach is correct and the result is RF it is mentioned at the table.
- $\mathcal{U}_{ir}$: If Y then this means that $\Delta_x(K \to K')^{\mathcal{U}_{ir}}(K) \sim K'$ i.e the approach is correct. Otherwise the cell is marked with N.

By definition the execution of a $\mathcal{U}_{ir}$ operation leaves the knowledge base in a redundancy free state, that is why at the column "$\mathcal{U}_{ir}$" we do mark the result as RF. Those pairs that have a N in the cells that concern correctness, constitute a proof (by counterexample) that they are *not correct*. For the rest pairs (those with a Y) we have to prove that they are *always* correct.

Theorem 1 For any pair of valid knowledge bases K and K' it holds:

$$\Delta_e(K \to K')^{\mathcal{U}_p}(K) \sim \Delta_{ed}(K \to K')^{\mathcal{U}_p}(K) \sim$$
$$\Delta_c(K \to K')^{\mathcal{U}_{ir}}(K) \sim \Delta_{dc}(K \to K')^{\mathcal{U}_{ir}}(K) \sim K'$$



(a) **Tree and Chain**						
Delta	$K \rightarrow K'$	$\mathcal{U}_p$	$\mathcal{U}_{ir}$	$K' \rightarrow K$	$\mathcal{U}_p$	$\mathcal{U}_{ir}$
Δ_e	$\{Add(C\ subClassOf\ B)$, $Del(C\ subClassOf\ A)\}$	Y, RF	Y	$\{Add(C\ subClassOf\ A)$, $Del(C\ subClassOf\ B)\}$	Y,RF	Y
Δ_c	$\{Add(C\ subClassOf\ B)\}$	Y	Y	$\{Del(C\ subClassOf\ B)\}$	**N**	Y
Δ_d	$\{Add(C\ subClassOf\ B)\}$	Y	Y	$\{Del(C\ subClassOf\ B)\}$	**N**	Y
Δ_{dc}	$\{Add(C\ subClassOf\ B)\}$	Y	Y	$\{Del(C\ subClassOf\ B)\}$	**N**	Y
Δ_{ed}	$\{Add(C\ subClassOf\ B)\}$	Y	Y	$\{Add(C\ subClassOf\ A)$, $Del(C\ subClassOf\ B)\}$	Y, RF	Y
(b) **Chain and Rooted DAG**						
Δ_e	$\{Add(C\ subClassOf\ A)$, $Add(D\ subClassOf\ B)$, $Del(C\ subClassOf\ B)\}$	Y, RF	Y	$\{Add(C\ subClassOf\ B)$, $Del(C\ subClassOf\ A)$, $Del(D\ subClassOf\ B)\}$	Y, RF	Y
Δ_c	$\{Del(C\ subClassOf\ B)\}$	**N**	Y	$\{Add(C\ subClassOf\ B)\}$	Y	Y
Δ_d	$\{Del(C\ subClassOf\ B)\}$	**N**	Y	$\{Add(C\ subClassOf\ B)\}$	Y	Y
Δ_{dc}	$\{Del(C\ subClassOf\ B)\}$	**N**	Y	$\{Add(C\ subClassOf\ B)\}$	Y	Y
Δ_{ed}	$\{Add(C\ subClassOf\ A)$, $Add(D\ subClassOf\ B)$, $Del(C\ subClassOf\ B)\}$	Y, RF	Y	$\{Add(C\ subClassOf\ B)\}$	Y	Y
(c) **Tree and DAG**						
Δ_e	$\{Add(C\ subClassOf\ D)$, $Del(A\ subClassOf\ D)\}$	Y, RF	**N**	$\{Add(A\ subClassOf\ D)$, $Del(C\ subClassOf\ D)\}$	Y, RF	Y
Δ_c	$\{Del(A\ subClassOf\ D)$, $Del(B\ subClassOf\ D)\}$	**N**	Y	$\{Add(A\ subClassOf\ D)$, $Add(B\ subClassOf\ D)\}$	Y	Y
Δ_d	$\{Del(A\ subClassOf\ D)\}$	**N**	**N**	$\{Add(A\ subClassOf\ D)\}$	Y	Y
Δ_{dc}	$\{Del(A\ subClassOf\ D)$, $Del(B\ subClassOf\ D)\}$	**N**	Y	$\{Add(A\ subClassOf\ D)\}$	Y	Y
Δ_{ed}	$\{Add(C\ subClassOf\ D)$, $Del(A\ subClassOf\ D)\}$	Y, RF	**N**	$\{Add(A\ subClassOf\ D)\}$	Y	Y
(d) **Forest and Chain**						
Δ_e	$\{Add(A\ subClassOf\ D)\}$	Y, RF	Y	$\{Del(A\ subClassOf\ D)\}$	Y, RF	**N**
Δ_c	$\{Add(A\ subClassOf\ D)$, $Add(B\ subClassOf\ D)$, $Add(C\ subClassOf\ D)\}$	Y	Y	$\{Del(A\ subClassOf\ D)$, $Del(B\ subClassOf\ D)$, $Del(C\ subClassOf\ D)\}$	Y, RF	Y
Δ_d	$\{Add(A\ subClassOf\ D)\}$	Y, RF	Y	$\{Del(A\ subClassOf\ D)\}$	Y, RF	**N**
Δ_{dc}	$\{Add(A\ subClassOf\ D)\}$	Y, RF	Y	$\{Del(A\ subClassOf\ D)$, $Del(B\ subClassOf\ D)$, $Del(C\ subClassOf\ D)\}$	Y, RF	Y
Δ_{ed}	$\{Add(A\ subClassOf\ D)\}$	Y, RF	Y	$\{Del(A\ subClassOf\ D)\}$	Y, RF	**N**

Table 2.2: Examples

Theorem 2 $\Delta_d(K \rightarrow K')^{\mathcal{U}_{ir}}(K) \sim K'$ iff (a) K is complete, or (b) $C(K) - K \subseteq C(K')$.

Theorem 3 $\Delta_c(K \rightarrow K')^{\mathcal{U}_p}(K) \sim K'$ iff K is complete.

The proof of the above theorems are presented at the Appendix. An interesting remark regarding Theorem 2 is that if $C(K) \subseteq C(K')$, then condition (b) holds. This means that we could use the pair $(\Delta_d, \mathcal{U}_{ir})$ in cases we know that $C(K) \subseteq C(K')$. For example if K is an ontology O and K' is an additional ontology O' that specializes O, then we are sure that $C(K) \subseteq C(K')$. In such cases we can use Δ_d (or alternatively Δ_{dc}) which give the smallest in size Deltas (Δ_{dc} returns the same Deltas).

2.4.3 Semantic Identify and Non-Redundancy of $(\Delta_x, \mathcal{U}_y)$-pairs

Prop. 8 If $K \sim K'$ then $\Delta_d(K \rightarrow K') = \Delta_c(K \rightarrow K') = \Delta_{dc}(K \rightarrow K') = \emptyset$.

This is property (P1) and its proof is trivial. Note that Δ_e is not included in Prop. 8 because even if $K \sim K'$, it may be $K = K'$, $K \subset K'$, $K' \subset K$, or $K \not\subseteq K'$ and $K' \not\subseteq K$. In the example of Figure 2.6 we get $\Delta_e(K \rightarrow K') = \{Add(C\ subClassOf\ A)\}$ although $K \sim K'$. It should be stressed that most of the existing differential functions [4, 41, 15] actually employ Δ_e, so they do not satisfy (P1).

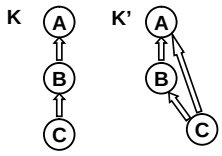


Figure 2.6: Two equivalent KBs

However, one can easily prove that: If K and K' are both redundancy free, and the knowledge bases considered have always a unique reduction, then $K \sim K' \Rightarrow \Delta_e(K \rightarrow K') = \emptyset$.

Prop. 9 If $K \sim K'$, $\{K, K'\} \subseteq \Psi$ and $RF(K), RF(K')$ then $\Delta_e(K \rightarrow K') = \emptyset$

Finally, it can be easily proved that if (a) $K' \subseteq K$ or (b) K and K' are both redundancy free, and the knowledge bases considered have always a unique reduction, then $K \sim K' \Rightarrow \Delta_{ed}(K \rightarrow K') = \emptyset$.

Prop. 10 If $K \sim K'$, and (a) $K' \subseteq K$ or (b) $\{K, K'\} \subseteq \Psi$ and $RF(K), RF(K')$ then $\Delta_{ed}(K \rightarrow K') = \emptyset$

It is worth mentioning that Δ_e and Δ_{ed} satisfy semantic identity under different conditions. Moreover at the case of Figure 2.5 (f) (i.e. $K = C(K')$) Δ_{ed} satisfies semantic identity but this is not the general case.

2.4.4 Reversibility of Δ_x

It is not hard to see that this property is satisfied by those differential functions whose formulas for add and del triples are symmetric So Δ_e, Δ_c and Δ_d satisfy (P3). Mention that the reverse of a differential function $Inv(\Delta_x(K \rightarrow K'))$ is correct only when the $\Delta_x(K' \rightarrow K)$ is correct (see section 2.4.2).

Prop. 11 For every pair of valid knowledge bases K, K' it holds:

$$Inv(\Delta_e(K \rightarrow K')) = \Delta_e(K' \rightarrow K)$$
$$Inv(\Delta_c(K \rightarrow K')) = \Delta_c(K' \rightarrow K)$$
$$Inv(\Delta_d(K \rightarrow K')) = \Delta_d(K' \rightarrow K)$$

Δ_{dc} and Δ_{ed} do not satisfy (P3). For example in the case (a) of Table 2.2 $\Delta_{ed}(K \rightarrow K') = \{Add(C\ subClassOf\ B)\}$, so $Inv(\Delta_{ed}(K \rightarrow K')) = \{Del(C\ subClassOf\ B)\}$, while $\Delta_{ed}(K' \rightarrow K) = \{Add(C\ subClassOf\ A), Del(C\ subClassOf\ B)\}$. This means that:

$$Inv(\Delta_{ed}(K \rightarrow K')) \neq \Delta_{ed}(K' \rightarrow K)$$

At the case (d) of Table 2.2 $\Delta_{dc}(K \rightarrow K') = \{Add(A\ subClassOf\ D)\}$, so $Inv(\Delta_{dc}(K \rightarrow K')) = \{Del(A\ subClassOf\ D)\}$, while $\Delta_{dc}(K' \rightarrow K) = \{Del(A\ subClassOf\ D), Del(B\ subClassOf\ D), Del(C\ subClassOf\ D)\}$. This means that:

$$Inv(\Delta_{dc}(K \rightarrow K')) \neq \Delta_{dc}(K' \rightarrow K)$$

Generally Δ_{dc} and Δ_{ed} do not satisfy reversibility, but if K' is complete (i.e. $K' = C(K')$) then they do satisfy reversibility. This happens because if $K' = C(K')$ then Δ_{dc} produces the same result as Δ_c and Δ_{ed} produces the same result as Δ_e (see Figure 2.5).

2.4.4.1 Reversibility and Delta size

As already mentioned it is beneficial for a differential function to be reversible. Unfortunately, not all the differential functions are reversible. So in order to be able to move forward and backward to versions we have to compute and store, apart from $\Delta_x(K \rightarrow K')$, also $\Delta_x(K' \rightarrow K)$. The fact that both deltas have to be stored requires more storage space and so the size of both deltas is crucial.

It has already been proved that only Δ_e and Δ_c are reversible in the general case. So we have to examine whether it is more space consuming to store the delta of Δ_e and Δ_c assuming only one direction (i.e. $\Delta_x(K \rightarrow K')$) or to store the delta of Δ_{ed} and Δ_{dc} at both directions (i.e. $\Delta_x(K \rightarrow K')$ and $\Delta_x(K' \rightarrow K)$).

Prop. 12 For any pair of knowledge bases K and K' it holds:

$$\begin{aligned} |\Delta_e(K \rightarrow K')| &\leq |\Delta_{ed}(K \rightarrow K')| + |\Delta_{ed}(K' \rightarrow K)| \\ |\Delta_c(K \rightarrow K')| &\leq |\Delta_{dc}(K \rightarrow K')| + |\Delta_{dc}(K' \rightarrow K)| \end{aligned}$$

2.4.5 Composability of Δ_x

The differential functions that always satisfy composability are Δ_e and Δ_c. Mention that the composability of a differential function $\Delta_x(K_1 \rightarrow K_2) \circ ... \circ \Delta_x(K_{n-1} \rightarrow K_n) = \Delta_x(K_1 \rightarrow K_n)$ is correct only when the $\Delta_x(K_1 \rightarrow K_n)$ is correct (see section 2.4.2)

Prop. 13 For any valid knowledge bases $K_1, K_2, K_3, ..., K_n$ it holds:

$$\begin{aligned} \Delta_e(K_1 \rightarrow K_2) \circ \Delta_e(K_2 \rightarrow K_3) \circ ... \circ \Delta_e(K_{n-1} \rightarrow K_n) &= \Delta_e(K_1 \rightarrow K_n) \\ \Delta_c(K_1 \rightarrow K_2) \circ \Delta_c(K_2 \rightarrow K_3) \circ ... \circ \Delta_c(K_{n-1} \rightarrow K_n) &= \Delta_c(K_1 \rightarrow K_n) \end{aligned}$$

It is not hard to see that this property is not satisfied by the Δ_d, Δ_{dc} and Δ_{ed} differential functions. For example, at the KBs of figure 2.7 (a) for Δ_d we have:

$$\begin{aligned}\Delta_d(K_1 \rightarrow K_2) &= \{Add(B\ subClassOf\ A)\}\\ \Delta_d(K_2 \rightarrow K_3) &= \{Del(B\ subClassOf\ A)\}\\ \Delta_d(K_1 \rightarrow K_3) &= \{Add(C\ subClassOf\ A)\}\end{aligned}$$

If we compose the first two deltas we get the empty result that is different from $\Delta_d(K_1 \rightarrow K_3)$:

$$\Delta_d(K_1 \rightarrow K_2) \circ \Delta_d(K_2 \rightarrow K_3) = \emptyset$$

The same result holds also for Δ_{dc}.

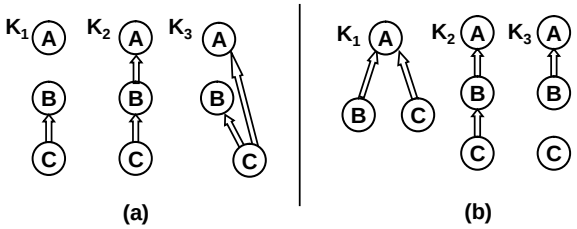


Figure 2.7: Composability of Δ_x

In the figure 2.7 (b) for Δ_{ed} we have:

$$\begin{aligned}\Delta_{ed}(K_1 \rightarrow K_2) &= \{Add(C\ subClassOf\ B)\}\\ \Delta_{ed}(K_2 \rightarrow K_3) &= \{Del(C\ subClassOf\ B)\}\\ \Delta_{ed}(K_1 \rightarrow K_3) &= \{Del(C\ subClassOf\ A)\}\end{aligned}$$

If we compose those two deltas we get the empty result that is different from $\Delta_{ed}(K_1 \rightarrow K_3)$:

$$\Delta_{ed}(K_1 \rightarrow K_2) \circ \Delta_{ed}(K_2 \rightarrow K_3) = \emptyset$$

Generally Δ_{dc} and Δ_{ed} do not satisfy reversibility, but if K' is complete (i.e. $K' = C(K')$) then they do satisfy composability. This happens because if $K' = C(K')$ then

Δ_{dc} produces the same result as Δ_c and Δ_{ed} produces the same result as Δ_e (see Figure 2.5).

2.4.6 Streaming Application/Execution of Deltas

Regarding the execution of deltas, so far we have assumed an execution mode as defined in Def. 5. An alterantive method would be to execute each operation in Δ^+ and Δ^- separately (according to the pre/post-conditions defined in Table 2.1). The rising question is whether these two alternative execution modes are equivalent. It is not hard to see that this is true for $\mathcal{U}_p$ semantics. However, as we shall see, the order of execution of change operations may affect the resulting KB under $\mathcal{U}_{ir}$ semantics. In particular the resulting KB may not satisfy all change operations returned by a differential function (see Def. 4), and we may get a wrong result. For instance, for the KBs of Table 2.2 (d) we get $\Delta_{dc}(K' \rightarrow K) =$

$$\{Del(A\ subClassOf\ D), Del(B\ subClassOf\ D), Del(C\ subClassOf\ D)\}$$

If the operations are executed in the order

$$\langle Del(A\ subClassOf\ D), Del(B\ subClassOf\ D), Del(C\ subClassOf\ D)\rangle$$

under $\mathcal{U}_{ir}$ semantics, then all of them will be satisfied and the result will be equivalent to K. Now consider the following execution order

$$\langle Del(B\ subClassOf\ D), Del(C\ subClassOf\ D), Del(A\ subClassOf\ D)\rangle$$

In this case the operation $Del(B\ subClassOf\ D)$ does not change the K as it requests the deletion of an inferred triple and according to $\mathcal{U}_{ir}$ semantics an inferred triple cannot be deleted. The same will happen with the operation $Del(C\ subClassOf\ D)$. Finally, the operation $Del(A\ subClassOf\ D)$ will be executed and will cause the addition of the triple $(B\ subClassOf\ D)$. It is obvious that the operation
$Del(B\ subClassOf\ D)$ is not satisfied by the resulting KB because it contains the triple $(B\ subClassOf\ D)$. We have just seen an example where the order of execution matters. The same problem occurs when K' contains a redundant triple e.g. $(B\ subClassOf\ D)$. A

similar situation is encountered with Δ_c and with Δ_d when K and K' are not redundancy free.

However we have proved that a multi-pass execution algorithm will eventually satisfy all operations and hence we will obtain the desired result. Specifically we can adopt the loop-based algorithm that is shown next.

Alg 1. **Execute**(K, M) where $M \subseteq \mathcal{S}$
(1) $UnSat = M$
(2) repeat
(3) $Sat = \emptyset$ //the set of all satisfied operations
(4) for i=1 to $|UnSat|$
(5) $u = UnSat[i]$ // gets the next operation
(6) $K_t = u^{\mathcal{U}_{ir}}(K)$ // applies the operation with $\mathcal{U}_{ir}$-semantics
(7) if K_t satisfies u then
(8) $Sat = Sat \cup \{u\}$
(9) $K = K_t$
(10) endif
(11) endfor
(12) $UnSat = UnSat - Sat$
(13) until $UnSat = \emptyset$

Obviously, this algorithm will not terminate unless all delete operations in M are satisfied. For the problem at hand, we have to prove that the execution algorithm always terminates if M has been derived from one of Δ_d, Δ_c and Δ_{dc}.

Prop. 14 For every $M \subseteq \mathcal{S}$ produced by Δ_d, Δ_{dc} or Δ_c, Alg. 1 terminates.

The proof is found at the Appendix. Below we describe in brief the crux of the proof. Let Y be the satisfiable deletions and Z the unsatisfiable deletions at any point during the execution of the algorithm (i.e. the sets Sat and $UnSat$ respectively). We have proved that whenever $|Y| = 0$ we also have $|Z| = 0$. This guarantees that the algorithm always terminates since all elements of M are satisfied.

Theorem 4 If $\{K, K'\} \subseteq \Psi$ then for every delta produced by Δ_d, Δ_{dc} and Δ_c there exists at least one sequence of change operations that when executed under $\mathcal{U}_{ir}$ the result is equivalent to the execution of the corresponding *set* of change operations under $\mathcal{U}_{ir}$. Alg. 1 produces such a sequence.

2.4.7 Summarizing the Results

The pairs that are always correct are: $(\Delta_e, \mathcal{U}_p)$, $(\Delta_{ed}, \mathcal{U}_p)$, $(\Delta_c, \mathcal{U}_{ir})$, $(\Delta_{dc}, \mathcal{U}_{ir})$. The pair $(\Delta_c, \mathcal{U}_p)$ is correct if K is complete, while the pair $(\Delta_d, \mathcal{U}_{ir})$ is correct in the cases specified in Theorem 2.

Δ	Sem.	Correctness	(P1)	(P2)	(P2.1)	(P3)	(P4)
Δ_e	$\mathcal{U}_p$	**Y**	see Prop. 9	N	Y	Y	Y
Δ_c	$\mathcal{U}_p$	Y if $K = C(K)$	Y	N	N	Y	Y
Δ_d	$\mathcal{U}_p$	N	-	-	-	-	-
Δ_{dc}	$\mathcal{U}_p$	N	-	-	-	-	-
Δ_{ed}	$\mathcal{U}_p$	**Y**	see Prop. 10	N	Y	Y if $K' = C(K')$	Y if $K' = C(K')$
Δ_e	$\mathcal{U}_{ir}$	N	-	-	-	-	-
Δ_c	$\mathcal{U}_{ir}$	**Y**	Y	Y	Y	Y	Y
Δ_d	$\mathcal{U}_{ir}$	see Th.2	Y	Y	Y	Y	N
Δ_{dc}	$\mathcal{U}_{ir}$	**Y**	Y	Y	Y	Y if $K' = C(K')$	Y if $K' = C(K')$
Δ_{ed}	$\mathcal{U}_{ir}$	N	-	-	-	-	-

Table 2.3: Synopsis

The differential functions that always satisfy *semantic identity* are: Δ_c, Δ_{dc} and Δ_d, while the functions Δ_e and Δ_{ed} satisfy semantic identity in the cases specified in Propositions 9 and 10 respectively. Concerning the *non-redundancy* criterion, the pairs that use $\mathcal{U}_{ir}$ as change operation semantics always satisfy non-redundancy, while the pairs $(\Delta_e, \mathcal{U}_p)$ and $(\Delta_{ed}, \mathcal{U}_p)$ satisfy only weaker non-redundancy. The pair $(\Delta_c, \mathcal{U}_p)$ does not satisfy neither non-redundancy nor weaker non-redundancy.

The differential functions that always satisfy *reversibility* are Δ_e, Δ_c and Δ_d, while Δ_{dc} and Δ_{ed} satisfy reversibility only if K' is complete (i.e. $K' = C(K')$). Finally, Δ_e and Δ_c always satisfy *composability*, while Δ_{dc} and Δ_{ed} satisfy composability only if K' is complete (i.e. $K' = C(K')$). Table 2.3 synopsizes the above results; note that the properties (P1)-(P4) are examined only for the pairs $(\Delta_x, \mathcal{U}_y)$ that are correct.

Chapter 3

Algorithm Specification and Implementation

3.1 Introduction

In Chapter 2 five differential functions are introduced and analyzed namely Delta Explicit (Δ_e), Delta Closure (Δ_c), Delta Dense (Δ_d) Delta Dense & Closure (Δ_{dc}) and Delta Explicit & Dense (Δ_{ed}). For each of them a formula that indicates the produced result is provided. In this chapter we will describe the algorithms used in order to implement those formulas.

Although some differential functions are defined over the closure of KBs, the algorithm that we present does not compute the closure of the corresponding KBs. However in several cases the algorithm has to compute all superclasses (resp. superproperties) of a particular class (resp. property). To further reduce the computational complexity the algorithm exploits a special labeling scheme (offered by the main memory representation of RDF models) that enables us to decide whether a particular class (resp. property) is directly or indirectly subsumed by another class (resp. property) in constant time.

3.2 Notations and Definitions

Below we introduce some basic notations that are used to describe the algorithm:

- c is used to denote an RDF class
- mc is used to denote an RDF metaclass
- p is used to denote an RDF property
- mp is used to denote an RDF metaproperty
- r is used to denote an RDF resource
- con is used to denote a container
- ci is used to denote a class instance
- tr is used to denote all the special Triples . The predicates of the special Triples are: $comment$, $isDefinedBy$, $label$, $seeAlso$ $subject$ (reification), $predicate$ (reification), $object$ (reification) and any $property\ Instance$

Let us now introduce some auxiliary notations that are used for the description of the algorithm:

$$\begin{aligned}
sup(c) &= \text{the direct superclasses of } c \\
supAll(c) &= \text{all the subclasses of } c \\
sup(p) &= \text{the direct superproperties of } p \\
supAll(p) &= \text{all the superproperties of } p \\
domain(p) &= \text{the domain of property } p \\
range(p) &= \text{the range of property } p \\
type(r) &= \text{the direct types of } r \\
typeAll(r) &= \text{all the types of } r \\
kind(con) &= \text{the kind of container } con \\
members(con) &= \text{all the member of container } con
\end{aligned}$$

Def. 7 A mapping $maps$ is a binary relation over the set of all possible URIs (i.e. $maps \subseteq URI \times URI$).

If two URIs u and u' are equivalent according to $maps$, we will write $maps(u, u')$.

In our current implementation (where a special naming scheme for URIs is adopted in order to allow having different versions of the same URI), we assume a default mapping between URI's. According to this mapping two URI's map if:

- their namespace parts are versions of the same URI and
- they have the same local name

For example the URIs "$http://myuri\tilde{}_\tilde{}v1\#A$" and "$http://myuri\tilde{}_\tilde{}v2\#A$" map, while the URIs "$http://myuri\tilde{}_\tilde{}v1\#A$" and "$http://otheruri\tilde{}_\tilde{}v1\#A$" do not map [1].

Apart from the default mapping it is possible to use a user provided mapping. According to a user provided mapping two URIs map if:

- their namespace parts are mapped at the user provided mapping and
- they have the same local name

For example, assume a user provided mapping $\langle http://myuri\#, http://otheruri\#\rangle$. Then the URIs "$http://myuri\#A$" and "$http://otheruri\#A$" map, while the URIs "$http://myuri\tilde{}_\tilde{}v1\#A$" and "$http://myuri\tilde{}_\tilde{}v2\#A$" do not map.

Obviously, if $u = u'$ then $maps(u, u')$. Overall, whether $maps(u, u')$ holds, depends on string equality, the particular versioning naming conventions and the user provided mappings.

The possible differential functions that can be used by the Algorithm are:

- Delta Explicit (Δ_e)
- Delta Closure (Δ_c)
- Delta Dense (Δ_d)
- Delta Dense & Closure (Δ_{dc})
- Delta Explicit & Dense (Δ_{dc})

[1] The symbol "˜_˜" is used to separate the version of the URI. This symbol can be used at a URI only for separating the version part of the URI.

3.3 Algorithm

3.3.1 Mapping Algorithm

This section describes the algorithm used to create the standard mapping i.e. the mapping between name/graph spaces that are versions of the same name/graph space.

Let $plain(uri)$ denote the part of the URI without the version. For example $plain(http://myuri\tilde{}_\tilde{}v1\#) = http://myuri\#$. The algorithm that creates the map table is the following:

Mapping Algorithm. Compute the mapping between two RDF KBs M, M'.

Input: Two RDF KBs: M, M'

Output: The mapping between the namespace URIs of the KBs M and M'.

```
for each namespaceURI ns in M
    for each namespaceURI ns' in M'
        if plain(ns)=plain(ns')
            mapping.add(ns,ns')
```

Two URIs u and u' map (i.e. $maps(u, u')$) if their namespace parts are mapped at the standard mapping and they have the same local name.

3.3.2 Diff Algorithm

This section describes the algorithm that computes the diff between two RDF KBs assuming a differential function.

Algorithm (Z-Diff) . Compute the Diff between two RDF KBs M, M'.

Input: Two KBs: M, M' a differential function d and a Mapping m

Output: The difference between M and M' assuming d .

```
Res=∅
for each class c in M{
    if (∃ c' | c' ∈ M' and  maps(c.uri, c'.uri))
        compareClasses(c,c',d)
```

else //class deleted

if ($d = \Delta_e$ or $d = \Delta_d$ or $d = \Delta_{ed}$)

Res=Res+$\{Del(c,$ subClassOf $, c_1) \mid c_1 \in sup(c)\}$

Res=Res+$\{Del(c,$ typeOf $, mc) \mid mc \in type(c)\}$

else //i.e. for $d = \Delta_c$ or $d = \Delta_{dc}$

Res=Res+$\{Del(c,$ subClassOf $, c_1) \mid c_1 \in supAll(c)\}$

Res=Res+$\{Del(c,$ typeOf $, mc) \mid mc \in typeAll(c)\}$

}

for each class c' in M' not matched { //class added

if (d=Δ_e or d=Δ_d or d=Δ_{dc} or $d = \Delta_{ed}$)

Res=Res+$\{Add(c',$ subClassOf $, c_1) \mid c_1 \in sup(c)\}$

Res=Res+$\{Add(c',$ typeOf $, mc) \mid mc \in type(c)\}$

else //i.e. for $d = \Delta_c$

Res=Res+$\{Add(c',$ subClassOf $, c_1) \mid c_1 \in supAll(c)\}$

Res=Res+$\{Add(c',$ typeOf $, mc) \mid mc \in typeAll(c)\}$

}

for each property p in M\{

if ($\exists\ p' \mid p' \in M'$ and $maps(p.uri, p'.uri)$)

compareProperties(p,p',d)

else //property deleted

Res=Res+$\{Del(p,$ domain $, c) \mid c \in domain(p)\}$

Res=Res+$\{Del(p,$ range $, c) \mid c \in range(p)\}$

if ($d = \Delta_e$ or $d = \Delta_d$ or $d = \Delta_{ed}$)

Res=Res+$\{Del(p,$ subPropertyOf $, p_1) \mid p_1 \in sup(p)\}$

Res=Res+$\{Del(p,$ typeOf $, mp) \mid mp \in type(p)\}$

else //i.e. for $d = \Delta_c$ or $d = \Delta_{dc}$

Res=Res+$\{Del(p,$ subPropertyOf $, p_1) \mid p_1 \in supAll(p)\}$

Res=Res+$\{Del(p,$ typeOf $, mp) \mid mp \in typeAll(p)\}$

}

for each property p' in M' not matched { //property added

Res=Res+$\{Add(p',$ domain $, c) \mid c \in domain(p')\}$

Res=Res+$\{Add(p',$ range $, c) \mid c \in range(p')\}$

if (d=Δ_e or d=Δ_d or d=Δ_{dc} or $d = \Delta_{ed}$)

Res=Res+$\{Add(p',$ subPropertyOf $, p_1) \mid p_1 \in sup(p)\}$

Res=Res+$\{Add(p',$ typeOf $, mp) \mid mp \in type(p)\}$

else //i.e. for $d = \Delta_c$

Res=Res+$\{Add(p',$ subPropertyOf $, p_1) \mid p_1 \in supAll(p)\}$

Res=Res+$\{Add(p',$ typeOf $, mp) \mid mp \in typeAll(p)\}$

}

for each class instance ci in M\{

if ($\exists\ ci' \mid ci' \in M'$ and $maps(ci.uri, ci'.uri)$)

compareResources(ci,ci',d)

else //class instance deleted

if ($d = \Delta_e$ or $d = \Delta_d$)

Res=Res+$\{Del(ci,$ typeOf $, mci) \mid mci \in type(ci)\}$

else //i.e. $d = \Delta_c$ or $d = \Delta_{dc}$

Res=Res+$\{Del(ci,$ typeOf $, mci) \mid mci \in typeAll(ci)\}$

}

for each class instance ci' in M' not matched\{ //class instance added

if (d=Δ_e or d=Δ_d or $d = \Delta_{dc}$ or $d = \Delta_{ed}$)

Res=Res+$\{Add(ci',$ typeOf $, mci) \mid mci \in type(ci)\}$

else //i.e. for $d = \Delta_c$

Res=Res+$\{Add(ci',$ typeOf $, mci) \mid mci \in typeAll(ci)\}$

}

for each container con in M\{

if ($\exists\ con' \mid con' \in M'$ and $maps(con.uri, con'.uri)$)

compareContainers(con,con',d)

else //class instance deleted

Res=Res+$\{Del(con,$ typeOf $, kind(con)\}$

Res=Res+$\{Del(con, li, m) \mid m \in members(con)\}$

}

for each container con' in M' not matched\{ //container added

Res=Res+$\{Add(con',$ typeOf $, kind(con)\}$

Res=Res+$\{Add(con', li, m) \mid m \in members(con)\}$

}

for each specialTriple tr in M

if ($\nexists\ tr' \mid tr' \in M'$ s.t. $maps(tr, tr')$) //triple deleted

Res=Res+$\{Del(tr)\}$

for each specialTriple tr' in M' not matched //triple added

Res=Res+$\{Add(tr')\}$

$\diamond$

The procedures **compareResources compareClasses** and **compareProperties** and **compareContainer** are described below.

procedure compareResources(r,r',d){

Res=$\emptyset$

if ($d = \Delta_e$)

for each $k \in type(r) - type(r')$

Res=Res+{Del(r,typeOf,k)}

for each $k \in type(r') - type(r)$

Res=Res+{Add(r',typeOf,k)}

if ($d = \Delta_c$)

for each $k \in typeAll(r)$

if ($k \notin type(r')$ and $k \notin supAll(type(r'))$ //using labels

Res=Res+{Del(r,typeOf,k)}

for each $k \in typeAll(r')$

if ($k \notin type(r)$ and $k \notin supAll(type(r))$ //using labels

Res=Res+{Add(r',typeOf,k)}

if ($d = \Delta_d$)

for each $k \in type(r)$

if ($k \notin type(r')$ and $k \notin supAll(type(r'))$ //using labels

Res=Res+{Del(r,typeOf,k)}

for each $k \in type(r')$

if ($k \notin type(r)$ and $k \notin supAll(type(r))$ //using labels

Res=Res+{Add(r',typeOf,k)}

if ($d = \Delta_{dc}$)

for each $k \in typeAll(r)$

if ($k \notin type(r')$ and $k \notin supAll(type(r'))$ //using labels

Res=Res+{Del(r,typeOf,k)}

for each $k \in type(r')$

if ($k \notin type(r)$ and $k \notin supAll(type(r))$ //using labels

Res=Res+{Add(r',typeOf,k)}

if ($d = \Delta_{ed}$)

for each $k \in type(r)$

if ($k \notin type(r')$ and $k \notin supAll(type(r'))$ //using labels

Res=Res+{Del(r,typeOf,k)}

for each $k \in type(r') - type(r)$

Res=Res+{Add(r',typeOf,k)}

}

procedure compareClasses(c,c',d)

Res=compareResources(c,c',d)

if ($d = \Delta_e$){

for each $k \in sup(c) - sup(c')$

Res=Res+{Del(c,subClassOf,k)}

for each $k \in sup(c') - sup(c)$

Res=Res+{Add(c',subClassOf,k)}

if ($d = \Delta_c$)

for each $k \in supAll(c)$

if $k \notin supAll(c')$ //using labels

Res=Res+{Del(c,subClassOf,k)}

for each $k \in supAll(c')$

if $k \notin supAll(c)$ //using labels

Res=Res+{Add(c',subClassOf,k)}

if ($d = \Delta_d$)

for each $k \in sup(c)$

if $k \notin supll(c')$ //using labels

Res=Res+{Del(c,subClassOf,k)}

for each $k \in sup(c')$

if $k \notin supAll(c)$ //using labels

Res=Res+{Add(c',subClassOf,k)}

if ($d = \Delta_{dc}$)

for each $k \in supAll(c)$

if $k \notin supAll(c')$ //using labels

Res=Res+{Del(c,subClassOf,k)}

for each $k \in sup(c')$

if $k \notin supAll(c)$ //using labels

Res=Res+{Add(c',subClassOf,k)}

if ($d = \Delta_{ed}$)

for each $k \in sup(c)$

if $k \notin supll(c')$ //using labels

Res=Res+{Del(c,subClassOf,k)}

for each $k \in sup(c') - sup(c)$

Res=Res+{Add(c',subClassOf,k)}

}

procedure compareProperties(p,p',d){

Res=compareResources(p,p',d)

if ($d = \Delta_e$){

for each $k \in sup(p) - sup(p')$

Res=Res+{Del(p,subPropertyOf,k)}

for each $k \in sup(p') - sup(p)$

Res=Res+{Add(p',subPropertyOf,k)}

if ($d = \Delta_c$)

for each $k \in supAll(p)$

if $k \notin supAll(p')$ //using labels

Res=Res+{Del(p,subPropertyOf,k)}

for each $k \in supAll(p')$

if $k \notin supAll(p)$ //using labels

Res=Res+{Add(p',subPropertyOf,k)}

if ($d = \Delta_d$)

for each $k \in sup(p)$

if $k \notin supll(p')$ //using labels

Res=Res+{Del(p,subPropertyOf,k)}

for each $k \in sup(p')$

if $k \notin supAll(p)$ //using labels

Res=Res+{Add(p',subPropertyOf,k)}

if ($d = \Delta_{dc}$)

for each $k \in supAll(p)$

if $k \notin supAll(p')$ //using labels

Res=Res+{Del(c,subPropertyOf,k)}

for each $k \in sup(p')$

if $k \notin supAll(p)$ //using labels

Res=Res+{Add(p',subPropertyOf,k)}

if ($d = \Delta_{ed}$)

for each $k \in sup(p)$

if $k \notin supll(p')$ //using labels

Res=Res+{Del(p,subPropertyOf,k)}

for each $k \in sup(p') - sup(p)$

Res=Res+{Add(p',subPropertyOf,k)}

for each $k \in domain(p) - domain(p')$

Res=Res+{Del(p,domain,k)}

for each $k \in domain(p') - domain(p)$

Res=Res+{Add(p',domain,k)}

for each $k \in range(p) - range(p')$

Res=Res+{Del(p,range,k)}

for each $k \in range(p') - range(p)$

Res=Res+{Add(p',range,k)}

}

procedure compareContainers(con,con',d){

Res=compareResources(p,p',d)

if $kind(con') = Seq$\{ //the sequence of the members matters

```
        for each m ∈ members(con), m′ ∈ members(con′)
            if members(con)[i] ≠ members(con′)[i]
                Res=Res+{Del(con,i,m)}
                Res=Res+{Add(con′,i,m′)}
    }else{ //the sequence of the members does not matter
        for each m ∈ members(con′) − members(con)
            Res=Res+{Add(con,li,m)}
        for each m ∈ members(con) − members(con′)
            Res=Res+{Del(con′,li,k)}
    }
}
```

3.3.3 RDF/S Delta Complexity

Let's consider the $\Delta_x(K_1 \rightarrow K_2)$ of two RDF/S KBs K_1 and K_2 for any $x \in \{e, c, d, dc, ed\}$. Let's assume the following sizes: $N_1 = |K_1|$, $N_2 = |K_2|$, $C_1 = |C(K_1)|$, $C_2 = |C(K_2)|$, while $N = \max(N_1, N_2)$, $C = \max(C_1, C_2)$, $M = max(N_2, C_1)$ and $L = max(N_1, C_2)$. Note that in the worst case $C_1 = N_1^2$ and $C_2 = N_2^2$. We study the time complexity of computing $\Delta_x(K_1 \rightarrow K_2)$ in the following three settings:

(i) All K_1, K_2, $C(K_1)$ and $C(K_2)$ are stored in corresponding hashtables. Thus, we can decide in $O(1)$ whether a triple t belongs to one of these collections.

(ii) Only K_1 and K_2 are stored in hashtables allowing us to decide in $O(1)$ whether a triple t belongs to a K_1 or K_2. The cost of computing $C(K_1)$ ($C(K_2)$) from (K_2) is in $O(L_1) = O(N_1^2)$ ($O(L_2) = O(N_2^2)$).

(iii) Only K_1 and K_2 are stored but a labeling scheme [10] is used[2] to decide in $O(\gamma)$ whether $t \in C(K)$, where the factor γ will be discussed later on.

Table 3.1 summarizes for each of the above settings the time complexity in order to compute Δ_x for any $x \in \{e, c, d, dc, ed\}$. At setting (i) Δ_e, Δ_d and Δ_{ed} are less expensive

[2]To simplify the presentation we ignore the cost of computing and storing the labels employed to encode subsumption hierarchies.

to compute than Δ_{dc}, while Δ_{dc} is less expensive than Δ_c. At setting (ii) Δ_e is less expensive to compute than Δ_{ed}, while Δ_{ed} is less expensive than Δ_d, Δ_{dc} and Δ_c. Finally, at setting (iii) Δ_e is less expensive to compute than Δ_d and Δ_{ed}, while Δ_d and Δ_{ed} are less expensive than Δ_{dc}. Δ_c is the most expensive to compute at this setting.

$K_1 \rightarrow K_2$	(i) all stored	(ii) only K's are stored	(iii) only K's and labels
Δ_e	$O(N)$	$O(N)$	$O(N)$
Δ_c	$O(C)$	$O(N^2)$	$O(N^2)$
Δ_d	$O(N)$	$O(N^2)$	$O(N\gamma)$
Δ_{dc}	$O(M)$	$O(N^2)$	$O(M\gamma)$
Δ_{ed}	$O(N)$	$O(N_2^2)$	$O(N\gamma)$

Table 3.1: Time complexity of the differential functions

Let's now discuss factor γ. If t is an inferred triple on subsumption or instantiation relationships, we need to check whether $t \in C(K)$. When labeling schemes [10] are used to encode subsumption relationships of RDF/S classes (or properties), subsumption checking (i.e. to check whether a class/property A is direct or indirect subclass/subproperty of a class/property B) can be performed in $O(1)$ just by comparing the involved class labels (no transitive closure computation is required). The same is true, for instance, checking instantiation (i.e. to check whether a resource r is direct or indirect instance of a class B). However, more than one labels per class is required when the subsumption hierarchy is a DAG (and not a tree). Furthermore, resources may be classified under several classes not necessarily related through subsumption relationships. In both cases more than one comparison need to be performed per subsumption or instance check. Since, in practice, the number of additional labels that need to be accessed is small, the γ factor captures the incurred small overhead.

3.4 Main Memory Implementation

The Main Memory Implementation is responsible for comparing two RDF Models as they are created by RDFSuite [18]. The result of the comparison is a "delta" (or "diff") describing the differences between the two models, the change(s) that should be applied

upon the first in order to get to the second.

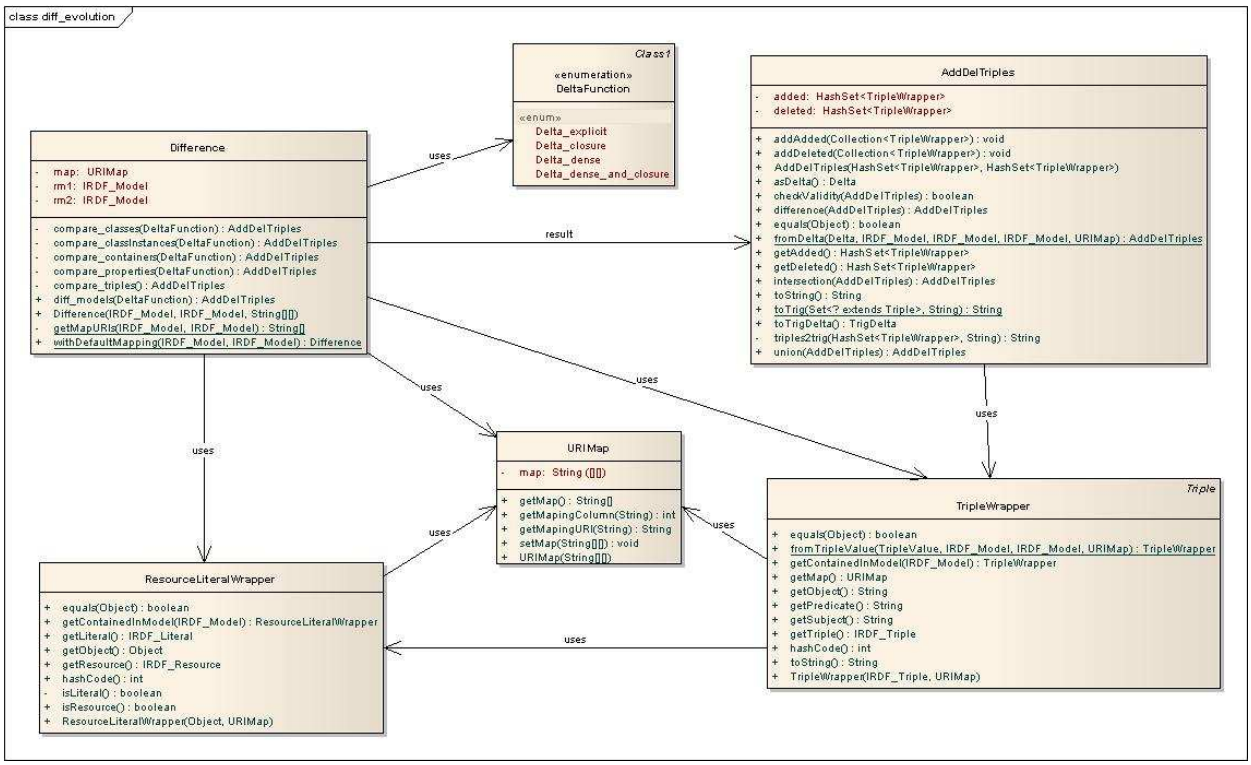


Figure 3.1: The UML diagram of the main memory implementation

The UML diagram of the main memory implementation is shown in Figure 3.1. The class URIMap implements the mapping. The data structure used to hold the mapping is a 2 dimension table with 2 columns and as many rows as the mappings. The restriction for the mapping is that one URI of the first model can map to only one URI of the second model. Furthermore, the mapping table holds mappings only between namespace URIs (i.e. URIs of the form $http://myuri\tilde{}_\tilde{}v1\#A$ are not allowed).

The main role of the classes ResourceLiteralWrapper and TripleWrapper is to implement different equality semantics than object equality. Those semantic are described in section 3.2. Triple equality is defined in a similar way. Two triples are equal if they are the same or if their subjects, predicates and objects are equal according to the equality defined by ResourceLiteralWrapper.

The class AddDelTriples hold the result computed by the Main Memory Diff Implementation. For this reason the class has two local variables HashSet one for the added

triples and one for the deleted.

Finally the class Difference computes the difference between two RDF Models. The variable DeltaFunction is an enumeration that indicates which of the five comparison functions is going to be used. The possible values for DeltFunction are: Delta explicit, Delta closure, Delta dense, Delta dense and closure and Delta explicit and dense. The function diff_models() uses other private functions to compare parts of the RDF Model: compare_classes(), compare_classInstances(), compare_properties() and compare_triples() . Those functions are private in order to allow the user to compare only RDF Models and not parts of them. The result of the comparison is returned as an AddDelTriples object. As default a standard mapping is used. The standard mapping maps URIs that are versions of the same URI.

3.5 Comparison Service

The SWKM platform (Figure 3.2) is a semantic management server-side stack, implemented in *Java* and *C++*. Its purpose is to provide its users scalable middleware services for managing voluminous representations of Semantic Web data (schemata and data expressed in RDF/S).

A major goal of the platform, which influences the design and implementation of almost all layers of the platform is to enable powerful and general declarative querying and update capabilities to the user.

The initial database schema is created by the *RSSDB*[3] component. RDF Schema information is stored into the database through RSSDB too, as currently RUL can only update RDF data instances, not schema information. Also, RSSDB offers bulk uploading of multiple data to the database, for any RDF-relational mapping, instead of having to issue multiple RUL statements to the RQL/RUL interpreter. Currently, PostgreSQL, an Object-Relational DBMS[4], is being used.

The SWKM offers eight services as an array of SOAP-enabled[5] Web Services. Those

[3]http://139.91.183.30:9090/RDF/RSSDB/
[4]http://www.postgresql.org/
[5]http://www.w3.org/TR/soap/

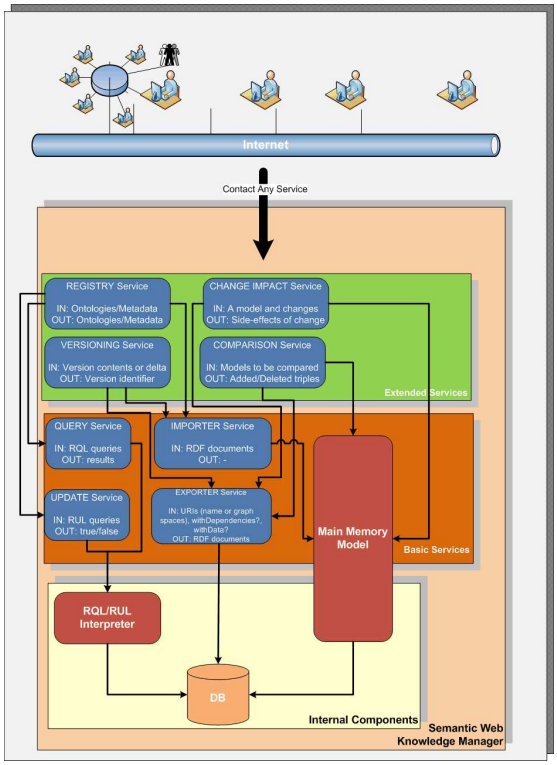


Figure 3.2: An architectural overview of the SWKM services

services are:

- **Import Service**. Is responsible for loading the contents of a valid and well formed name or graph space.
- **Exporter Service**. Is responsible for dumping into a byte sequence (in RDF/XML serialization or TRIG triple-based formats) the contents of the name or graph spaces given as input.
- **Query Service**. Is responsible for executing RQL queries and return the results in an RDF/XML or Trig serialization.
- **Update Service**. Is responsible for executing RUL updates involving one or several name or graph spaces.
- **Change Service**. Is responsible for determining the changes that should occur on a name or graph space in response to a change request [27].

- **Registry Service**. Is responsible for recording and managing metadata information about ontologies, schemas or namespaces stored in the repository.
- **Versioning Service**. Is responsible for constructing a new persistent version of a name or graph space already stored in the repository.
- **Comparison Service**.

The Comparison Service (Figure 3.3) is responsible for comparing two collections of name or graph spaces already stored in the repository and compute their delta in an appropriate form according to the selected differential function. For the comparison the Main Memory Implementation described at Section 3.4 is used.

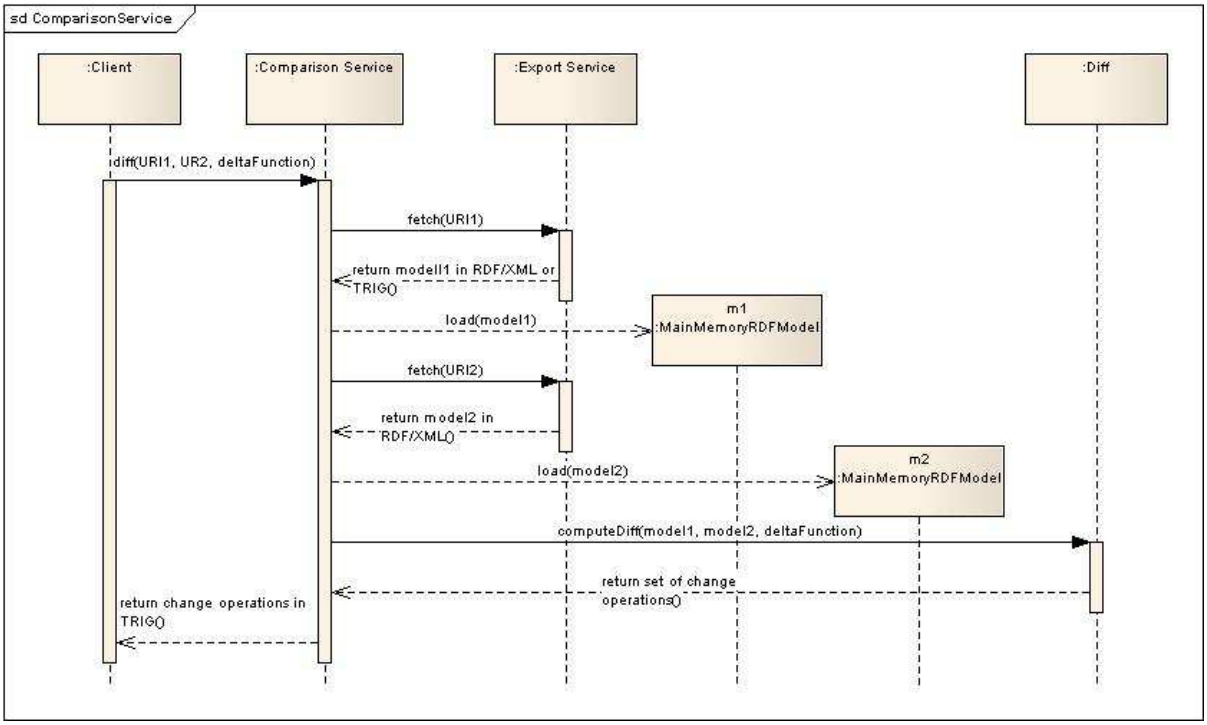


Figure 3.3: The Comparison service sequence diagram

Each collection of name or graph spaces is an array of strings, each string containing the URI of a name or graph space. It should be emphasized that the comparison is not performed only upon the name and graph spaces in the input, but also upon the name and graph spaces that they depend on. For example, if at the input exists only the namespace "$http://myuri\#$" but contains the triple $(http://myuri\#A, subclassof, http:$

//otheruri#B) (see Figure 3.4) then also the namespace "*http* : *//otheruri#*" will be used in the comparison. The deltaFunction parameter indicates the type of the differential function to be used in the comparison.

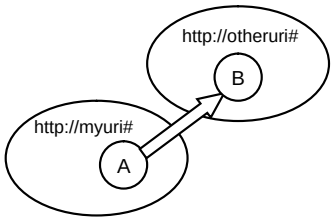


Figure 3.4: Two dependent namespaces

The output of the above operation is a pair of strings representing the delta between the two models. In particular, the first string of the pair represents the RDF triples that exist in the second model but not in the first (i.e. added triples), whereas the second represents the triples that exist in the first but not in the second (i.e. deleted triples). This way, the delta can be viewed as an update request which when applied to the first model, will (should) result to the second. Both strings should encode those triples in TRIG format.

The programmatic interface to this service is:

```
element diff {
    element URI1 { text }*,
    element URI2 { text }*,
    element deltaFunction { text },
    element dbSettings { dbSettingsElement }?
}
element diffResponse {
    element delta { deltaElement }
}
```

3.6 Experimental Evaluation

In this Section we experimentally measure the time required to compare real and synthetic RDF/S KBs of variable size and structure. In addition, we are interested in

comparing the size (i.e. triples to be added or deleted) of the produced Deltas with respect to the different application setting presented in Section 1.3.1 (i.e. forward/backward, reverse deltas). It is worth mentioning that our experimental findings confirm the respective analytical evaluation (see Section 2.2 and 3.3.3) of the five differential functions while revealing useful information about the cases where such an analysis is not available.

All experiments were carried out in a PC with processor Intel Core2 Quad 2.4 Ghz, 2 GB Ram, running Windows Vista. We assume that both KBs have been loaded in main memory using a labeling schema for encoding transitive closures and thus net differential times are reported.

3.6.1 Testbed

Our testbed comprises the following datasets:

- **Real Data Set:** We used the RDF/S dumps from the Gene Ontology (GO) project[6] as a representative of large-scale evolving Semantic Web data. GO terms (i.e. genes) are exported in a simple RDF/S schema containing only one meta-class, instance of which are all the 53,574 classes representing the GO terms in the latest version of the dump. Subsumption relationships between classes are represented by user-defined properties (i.e. not by "subClassof" properties). GO employs in total 11 properties to describe genes. The GO RDF/S KB does not contain any redundant triples while it employs heavily blank nodes (e.g. to represent structured values as tuples). Specifically, 28,171 of the 53,574 instance resources are blank nodes. Additionally, there are 400,999 property instances, most of which connect a blank node to a literal value. It is worth noticing that GO curators usually reclassify terms as obsolete and thus they are never actually deleted; this is done so that existing biological annotations do not have dangling reference during the time lag between the term being made obsolete and the reannotation of the entity.

- **Synthetic Data Set:** As the GO RDF/S KB has rather a simple structure (i.e. without a significantly large number of inferred triples) we have also created and used

[6] http://www.geneontology.org/

two synthetic data sets. Specifically, using the synthetic KB generator described in [37], we created a sequence of four KBs, $K_1, \ldots, K_4$, with 100, 200, 300, and 400 classes respectively. The number of properties $|E|$ in each KB is $3 * N$ where N is the number of classes. In the **simple** case, for each class 10 instances were created, while in the **complex** one 100 instances were created. In both data sets, for each property 10 property instances were created among randomly selected instances of the corresponding domain and range class. To simulate their evolution, we adopted a naming convention such that all classes, properties, and their instances in K_i are also present in K_{i+1} for each i=1..3. However, their subsumption structure may differ: classes which are higher in the subsumption hierarchy in version K_i may be found at a lower level in version K_{i+1}. This results in both additions and deletions of explicit triples (i.e. "subClassof" properties). The depth of the class subsumption hierarchy in each schema is 7, while redundant triples are asserted.

In order to capture the effect of poor vs rich subsumption structures in RDF/S KBs, we introduce the following metric regarding their inference potential.

Def. 8 *The* inference strength *of a knowledge base K, denoted by $is(K)$, is defined as:*

$$is(K) = \frac{|C(K)| - |K|}{|K|}$$

Clearly, if $K = C(K)$ then $is(K) = 0$. As one would expect the greater this factor is, the greater the number of new inferred triples in $C(K)$ w.r.t. K.

3.6.2 Experimental Results

For each compared RDF/S KB version we report its size and inference strength as well as the time (in seconds) required to compute the Deltas as well as their sizes (in triples). Specifically, for each Delta we report in parentheses the added and deleted triples ($(|\Delta^+|, |\Delta^-|)$) and then their total size ($|\Delta|$).

Table 3.2 and 3.3 report respectively, the Delta computing time and size for non-successive versions of the Biological data set. Note that Deltas are computed for both forward and backward versions of RDF/S KBs. As we can observe, the produced Deltas

exhibit almost similar sizes: Δ_e, Δ_{ed}, Δ_d and Δ_{dc} differ on at most 2 change operations. Only the size of Δ_c is bigger by at most 63 triples. This is due to the fact that the is factor is very small in this dataset.

K			K'			Time of ($\Delta(K \rightarrow K')$)				
KB	Triples	$is(K)$	KB	Triples	$is(K')$	Δ_e	Δ_{ed}	Δ_c	Δ_d	Δ_{dc}
K_{b1}	2898	0.229	K_{b2}	2964	0.229	0.281	0.289	0.329	0.296	0.312
K_{b3}	4493	0.225	K_{b4}	4816	0.223	0.594	0.603	0.765	0.625	0.648
K_{b5}	5994	0.218	K_{b6}	6068	0.218	0.967	0.973	1.046	0.975	0.998
K_{b7}	7340	0.205	K_{b8}	7399	0.205	1.119	1.128	1.354	1.133	1.149
K_{b9}	9028	0.196	K_{b10}	9217	0.196	1.665	1.682	1.763	1.698	1.718
K_{b11}	10806	0.182	K_{b12}	12772	0.182	1.973	2.021	2.090	1.979	2.042
K_{b13}	11680	0.178	K_{b14}	11779	0.178	2.181	2.196	2.300	2.187	2.249

Table 3.2: Delta time for Biological Data

Tables 3.4 and 3.5 report respectively, the Delta computing time and size for four consequent versions of the simple Synthetic data set. We can easily observe the significant divergences on Delta sizes. Specifically, on average the deltas produced by Δ_c are 63% bigger than those of Δ_e. The deltas of Δ_{dc} are 16% bigger than those of Δ_e. Note however, that Δ_c Δ_d and Δ_{dc} satisfy properties like semantic identity and also non-redundancy when used with $\mathcal{U}_{ir}$-semantics. The delta of Δ_d is 1% smaller than the size of Δ_e and the delta of Δ_{ed} is 0.5% smaller than the size of Δ_e. The minor differences in $|\Delta_e|$, $|\Delta_{ed}|$ and $|\Delta_d|$ sizes are due to the fact that most changes occur at the explicit graph i.e. only few triples that are inferred by K became explicit in K' and vice versa. On the other hand, the significant divergences between $|\Delta_e|$ and $|\Delta_c|$ sizes are due to changes occurring higher at the subsumption hierarchy. For the same reason, the deletions reported by Δ_{dc} are more than those reported by Δ_e, Δ_{ed} and Δ_d. In general, for Δ_e, Δ_{ed} and Δ_d all kinds of changes affect in the same way the delta size, while for Δ_c additions or deletions that occur highly at the subsumption hierarchy affect the delta size more. Finally, for Δ_{dc} all additions have the same impact, but deletions that occur highly at the subsumption hierarchy affect more the size of the produced delta.

Our experimental findings are summarized in Figure 3.5 which confirms the ordering of Delta sizes illustrated in Figure 2.5. Recall that Δ_e produces a big in size result if $K = C(K')$, while Δ_e and Δ_{ed} produce a big in size result if $K' = C(K')$.

Tables 3.6 and 3.7 report respectively, the Delta computing time and size for four successive versions of the complex Synthetic data set with a bigger inference strength

Size of $\Delta(K \rightarrow K')$						
K	K'	Δ_e	Δ_{ed}	Δ_c	Δ_d	Δ_{dc}
K_{b1}	K_{b2}	(79, 13) 92	(79, 13) 92	(94, 13) 107	(79, 13) 92	(79, 13) 92
K_{b3}	K_{b4}	(425, 102) 527	(425, 102) 527	(487, 103) 590	(425, 102) 527	(425, 103) 528
K_{b5}	K_{b6}	(92, 18) 110	(92, 18) 110	(107, 18) 125	(92, 18) 110	(92, 18) 110
K_{b7}	K_{b8}	(70, 11) 81	(70, 11) 81	(79, 11) 90	(70, 11) 81	(70, 11) 81
K_{b9}	K_{b10}	(250, 61) 311	(250, 61) 311	(286, 63) 349	(250, 61) 311	(250, 63) 313
K_{b11}	K_{b12}	(67, 4) 71	(67, 4) 71	(79, 4) 83	(67, 4) 71	(67, 4) 71
K_{b13}	K_{b14}	(117, 18) 135	(117, 18) 135	(134, 18) 152	(117, 18) 135	(117, 18) 135
Size of $\Delta(K' \rightarrow K)$						
K'	K	Δ_e	Δ_{ed}	Δ_c	Δ_d	Δ_{dc}
K_{b14}	K_{b13}	(18, 117) 135	(18, 117) 135	(18, 134) 152	(18, 117) 135	(18, 134) 152
K_{b12}	K_{b11}	(4, 67) 71	(4, 67) 71	(4, 79) 83	(4, 67) 71	(4, 79) 83
K_{b10}	K_{b9}	(61, 250) 311	(61, 250) 311	(63, 286) 349	(61, 250) 311	(61, 286) 347
K_{b8}	K_{b7}	(11, 70) 81	(11, 70) 81	(11, 79) 90	(11, 70) 81	(11, 79) 90
K_{b6}	K_{b5}	(18, 92) 110	(18, 92) 110	(18, 107) 125	(18, 92) 110	(18, 107) 125
K_{b4}	K_{b3}	(102, 425) 527	(102, 425) 527	(103, 487) 590	(102, 425) 527	(102, 487) 589
K_{b2}	K_{b1}	(13, 79) 92	(13, 79) 92	(13, 94) 107	(13, 79) 92	(13, 94) 107

Table 3.3: Delta size for Biological Data

K			K'			Time of $(\Delta(K \rightarrow K'))$				
KB	Triples	$is(K)$	KB	Triples	$is(K')$	Δ_e	Δ_{ed}	Δ_c	Δ_d	Δ_{dc}
K_{s1}	5,162	0.821	K_{s2}	10,267	0.789	0.570	0.583	0.952	0.594	0.718
K_{s2}	10,267	0.789	K_{s3}	15,389	0.806	0.889	0.903	1.499	1.187	1.388
K_{s3}	15,389	0.806	K_{s4}	20,460	0.826	1.296	1.352	2.217	1.623	1.702
K_{s1}	5,162	0.821	K_{s4}	20,460	0.826	1.077	1.088	1.624	1.109	1.092
K_{s4}	20,460	0.826	$C(K_{s4})$	37,369	0	1.225	1.358	1.782	1.462	1.639

Table 3.4: Delta time for simple Synthetic Data

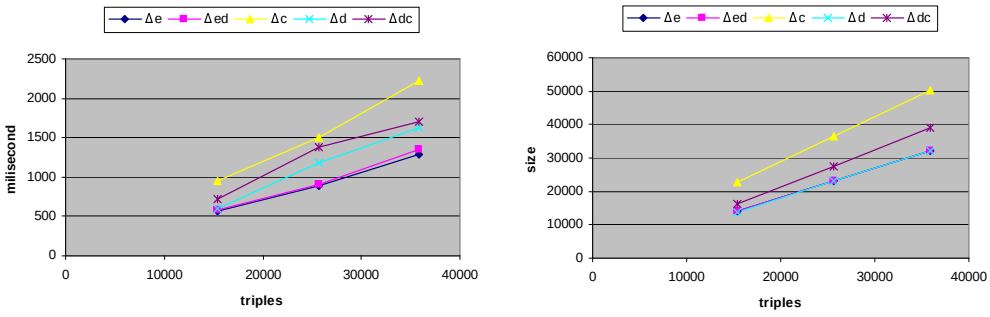


Figure 3.5: Delta time and size for simple Synthetic Data

Size of $\Delta(K \rightarrow K')$						
K	K'	Δ_e	Δ_{ed}	Δ_c	Δ_d	Δ_{dc}
K_{s1}	K_{s2}	(9,490, 4,509) 13,999	(9,490, 4,434) 13,924	(15,905, 6,932) 22,837	(9,430, 4,434) 13,864	(9,430, 6,932) 16,362
K_{s2}	K_{s3}	(14,089, 9,096) 23,185	(14,089, 9,022) 23,111	(22,937, 13,512) 36,449	(14,041, 9,022) 23,063	(14,041, 13,512) 27,553
K_{s3}	K_{s4}	(18,579, 13,635) 32,214	(18,579, 13,548) 32,127	(29,993, 20,422) 50,415	(18,519, 13,548) 32,067	(18,519, 20,422) 38,941
K_{s1}	K_{s4}	(19,461, 4,543) 24,004	(19,461, 4,495) 23,956	(34,960, 6,991) 41,951	(19,442, 4,495) 23,937	(19,442, 6,991) 26,433
K_{s4}	$C(K_{s4})$	(17,409, 0) 17,409	(17,409, 0) 17,409	(0, 0) 0	(0, 0) 0	(0, 0) 0
Size of $\Delta(K' \rightarrow K)$						
K'	K	Δ_e	Δ_{ed}	Δ_c	Δ_d	Δ_{dc}
$K_{C(s4)}$	K_{s4}	(0, 17,409) 17,409	(0, 0) 0	(0, 0) 0	(0, 0) 0	(0, 0) 0
K_{s4}	K_{s1}	(4,543, 19,461) 24,004	(4,543, 19,442) 23,985	(6,991, 34,960) 41,951	(4,495, 19,442) 23,937	(4,495, 34,960) 39,455
K_{s4}	K_{s3}	(13,635, 18,579) 32,214	(13,635, 18,519) 32,154	(20,422, 29,993) 50,415	(13,548, 18,519) 32,067	(13,548, 29,993) 43,541
K_{s3}	K_{s2}	(9,096, 14,089) 23,185	(9,096, 14,041) 23,137	(13,512, 22,937) 36,449	(9,022, 14,041) 23,063	(9,022, 22,937) 31,959
K_{s2}	K_{s1}	(4,509, 9490) 13,999	(4,509, 9,430) 13,939	(6,932, 15905) 22,837	(4,434, 9430) 13,864	(4,434, 15,905) 20,339

Table 3.5: Delta size for simple Synthetic Data

than the previous ones. In this case, significant differences in Delta sizes are observed. Specifically, on average Δ_c is 212% bigger than Δ_e. Δ_{dc} is 57% bigger than Δ_e. Δ_d is 1.3% smaller than Δ_e. Δ_{ed} is 0.8% smaller than Δ_e. Significant divergences also observed in Delta computing time.

K			K'			Time of $(\Delta(K \rightarrow K'))$				
KB	Triples	$is(K)$	KB	Triples	$is(K')$	Δ_e	Δ_{ed}	Δ_c	Δ_d	Δ_{dc}
K_{l1}	14,250	2.970	K_{l2}	28,355	2.868	1.333	1.339	5.382	1.844	2.590
K_{l2}	28,355	2.868	K_{l3}	42,477	2.923	3.266	2.511	9.302	3.413	5.201
K_{l3}	42,477	2.923	K_{l4}	56,546	2.987	3.443	3.643	13.442	5.071	7.329
K_{l1}	14,250	2.970	K_{l4}	56,546	2.987	2.131	2.192	7.741	2.205	3.405

Table 3.6: Delta time for Complex Synthetic Data

The experimental results of Figures 3.5 and 3.6 confirm the time complexity of Deltas presented in Table 3.1. We can observe that the execution time for Δ_c is always greater than the other ones. Furthermore, the execution times of Δ_e, Δ_d and Δ_{ed} are very close and smaller than Δ_{dc}.

Finally, Figure 3.7 (a) depicts how the inference strength affects the Delta sizes, while Figure 3.7 (b) illustrates how the inference strength affects the size of Deltas required in order to be able to move both forward and backward to KB versions. Since Δ_e and Δ_c are

Size of $\Delta(K \to K')$						
K	K'	Δ_e	Δ_{ed}	Δ_c	Δ_d	Δ_{dc}
K_{l1}	K_{l2}	(27,182, 13,077) 40,259	(27,182, 12,372) 39,547	(90,207, 37,100) 127,307	(26,762, 12,372) 39,134	(26,762, 37,100) 63,862
K_{l2}	K_{l3}	(41,270, 27,148) 68,418	(41,270, 26,804) 68,074	(128,328, 71,344) 199,672	(40,952, 26,804) 67,756	(40,952, 71,344) 112,296
K_{l3}	K_{l4}	(54,975, 40,906) 95,881	(54,975, 40,549) 95,524	(167,279, 108,443) 275,722	(54,645, 40,549) 95,194	(54,645, 108,443) 163,088
K_{l1}	K_{l4}	(55,407, 13,111) 68,518	(55,407, 12,703) 68,110	(206,266, 37,339) 243,605	(55,388, 12,703) 68,091	(55,388, 37,339) 92,727
Size of $\Delta(K' \to K)$						
K'	K	Δ_e	Δ_{ed}	Δ_c	Δ_d	Δ_{dc}
K_{l4}	K_{l1}	(13,111, 55,407) 68,518	(13,111, 55,388) 68,499	(37,339, 206,266) 243,605	(12,703, 55,388) 68,091	(12,703, 206,266) 218,969
K_{l4}	K_{l3}	(40,906, 54,975) 95,881	(40,906, 54,645) 95,551	(108,443, 167,279) 275,722	(40,549, 54645) 95,194	(40,549, 167,279) 207,828
K_{l3}	K_{l2}	(27,148, 41,270) 68,418	(27,148, 40,952) 68,100	(71,344, 128,328) 199,672	(26,804, 40,952) 67,756	(26,804, 128,328) 155,132
K_{l2}	K_{l1}	(13,077, 27,182) 40,259	(13,077, 26,762) 39,839	(37,100, 90,207) 127,307	(12,372, 26,762) 39,134	(12,372, 90,207) 102,579

Table 3.7: Delta size for the Complex Synthetic Data

reversible we report only the size of delta in one direction (i.e. $|\Delta(K \to K')|$), while for Δ_{dc} and Δ_{ed} we report the bisectional Delta size: $|\Delta(K \to K')| + |\Delta(K' \to K)|$. As we can see in Figure 3.7 (a), the less the inference strength is, the less the delta computing time and size differ.

Last but not least, in the case of handling both forward and backward versions, Δ_e appear to be the cheaper in size solution, followed by Δ_{ed}, then by Δ_c and finally by Δ_{dc}. This relationship is not captured by our analytical study (see Prop. 12).

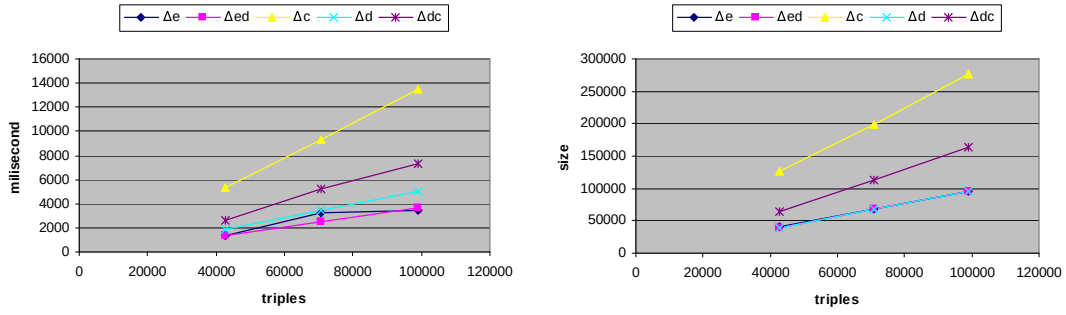


Figure 3.6: Delta time and size for Complex Synthetic Data

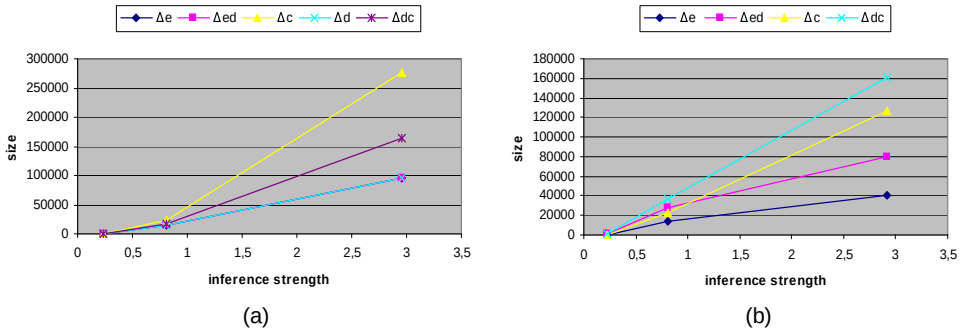


Figure 3.7: (a) Size of Forward Deltas (b) Size of Forward/Backward Deltas

Chapter 4

Related Work

This section examines the state of the art in tools that compare RDF graphs of the Semantic web.

4.1 Introduction

Several non text-based tools have been recently developed for comparing RDF graphs produced autonomously on the SW as for example:

- Ontoview [24]. Is an ontology management system, able to compare two ontology versions and highlight their differences.
- PromptDiff [32, 33, 31]. Is an ontology-versioning environment, that includes a version-comparison algorithm (based on heuristic matchers)
- SemVersion [41]. Proposes two Diff algorithms one *structure-based* and one *semantic-aware*
- RDF_Utils [15]. Introduce the notion of RDF molecules as the finest components to be used when comparing RDF graphs.
- CWM of w3c [4]. Is a general-purpose semantic web data processing tool which can compare two RDF files. It uses a functional or inverse functional properties to identify a blank nodes.

- Jena [1]. Is a Java framework for building Semantic Web applications. It provides a tool for checking isomorphism between two RDF graphs.

- Powl [3]. Is a web based ontology management tool that tracks the editing actions that are made using the system.

Existing RDF comparison tools have not yet focused on the size of the produced Deltas, a very important aspect for building versioning services over SW repositories [36]. Furthermore, the output of these tools is exploited by humans, and thus an intuitive presentation of the comparison results (and other related issues) has received considerable attention.

Finally, tracking the evolution of ontologies when changes are preformed in more controlled environments (e.g. collaborative authoring tools) has been addressed in [25, 34, 42].

4.2 Ontoview

OntoView [24] is a web-based system[2] inspired by CVS [6] that helps users to manage changes in ontologies. OntoView stores the contents of the versions, metadata, conceptual relations between constructs in the ontologies and the transformations between them. The internal version management is partly based on change specifications and the versions of ontologies themselves, but also uses additional human input about the meta-data and types of changes (as described below). It allows users to differentiate between ontologies at a conceptual level and to export the differences as adaptations or transformations.

OntoView provides a web "diff" view for comparing two versions of an ontology (see Figure 4.1). The comparison function is inspired by UNIX diff, but the implementation is quite different. This is a line-based tool, where the order of text is significant. So to produce a meaningful difference for ontologies where there is no inherent ordering, the ontology is canonicalized at the syntactic level before being given to the diff tool. In this web view, the user can characterize each difference, as explained previously.

[1] http://jena.sourceforge.net/

[2] A web demo is available here: http://test.ontoview.org/ but is currently not functional

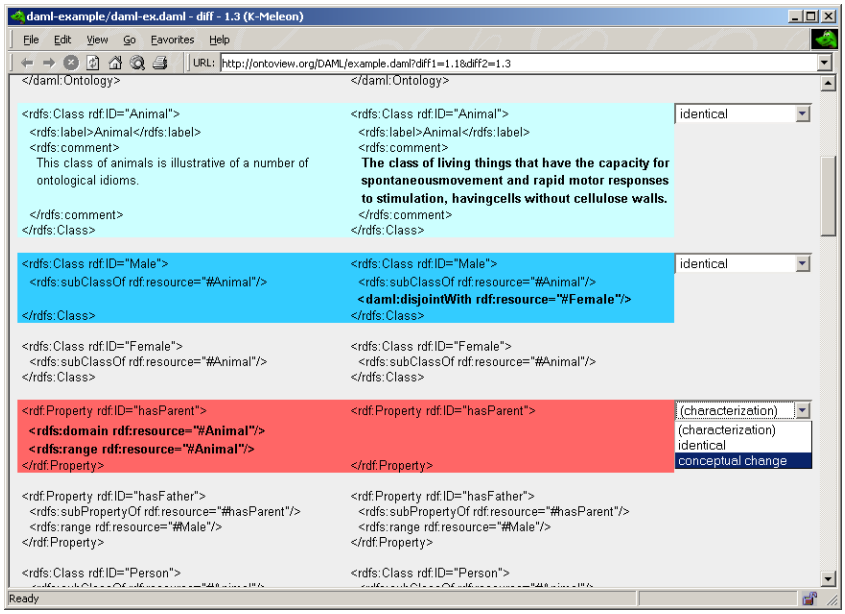


Figure 4.1: Comparing two ontologies in OntoView

The comparison function used by the Ontoview distinguishes between the following types of change:

- Non-logical change (conceptual change). A change at the natural language defintion. e.g. changes in the rdfs:label of a concept or property, or in a comment inside a definition.
- Logical definition change (explication change). This is a change in the definition of a concept or property that affects its formal semantics. Examples of such changes are alterations of subClassOf, domain, or range statements. Additions or deletions of local property restrictions in a class are also logical changes.
- Identifier change. This is the case when a concept or property is given a new identifier, i.e. a renaming.
- Addition of definitions.
- Deletion of definitions.

Most of these changes can be detected completely automatically, except for the identifier change, because this change is not distinguishable from a subsequent deletion and

addition of a simple definition. In this case, the system uses the location of the definition in the file as a heuristic to determine whether it is an identifier change or not.

4.2.1 Rules for Changes

The algorithm uses the fact that the RDF data model underlies a number of popular ontology languages, including RDF Schema and DAML+OIL. The RDF data model basically consists of triples of the form $\langle subject, predicate, object \rangle$, which can be linked by using the object of one triple as the subject of another. There are several syntaxes available for RDF statement, but they all boil down to the same data model. A set of related RDF statements can be represented as a graph with nodes and edges.

The algorithm used by Ontoview that detects changes is the following. First, it splits the document at the first level of the XML document. This groups the statements by their intended "definition". The definitions are then parsed into RDF triples, which results in a set of small graphs. Each of these graphs represent a specific definition of a concept or a property, and each graph can be identified with the identifier of the concept or the property that it represents.

Then, the algorithm locates for each graph in the new version the corresponding graph in the previous version of the ontology. Those sets of graphs are then checked according to a number of rules. Those rules specify the "required" changes in the triples set (i.e., the graph) for a specific type of change.

The rules have the following format:

IF exist:old

$\langle A, Y, Z \rangle *$

exist:new

$\langle X, Y, Z \rangle *$

not-exist:new

$\langle X, Y, Z \rangle *$

THEN change-type A

The rules are specific for a particular RDF-based ontology language because they encode the interpretation of the semantics of the language for which they are intended. For another language other rules would have been necessary to specify other differences in interpretation. The semantics of the language are thus encoded in the rules. The mechanism relies on the "materialization" of all rdf:type statements that are encoded in the ontology. In other words, the closure of the RDF triples according to the used ontology language has to be computed.

The application of the rules thus has to be preceded by the materialization of the superclass- and superproperty hierarchies in the ontology. For this materialization, the entailment and closure rules in the RDF Model Theory can be used.

4.3 Promptdiff

`Prompt` is an ontology-management framework that brings together different ontology-management tools and provides an infrastructure for other related tools. The key components of the framework are: `iPrompt` an interactive ontology-merging tool, `AnchorPrompt` a graph-based tool for finding related concepts in different ontologies, `Protege` a tool that provides access to a library of ontologies, giving users meta-information about an ontology and `PromptDiff`.

PromptDiff is an ontology-versioning tools that determines what has changed between two versions. It finds a structural diff between versions i.e. compares the structure of ontology versions and not their text serialization.

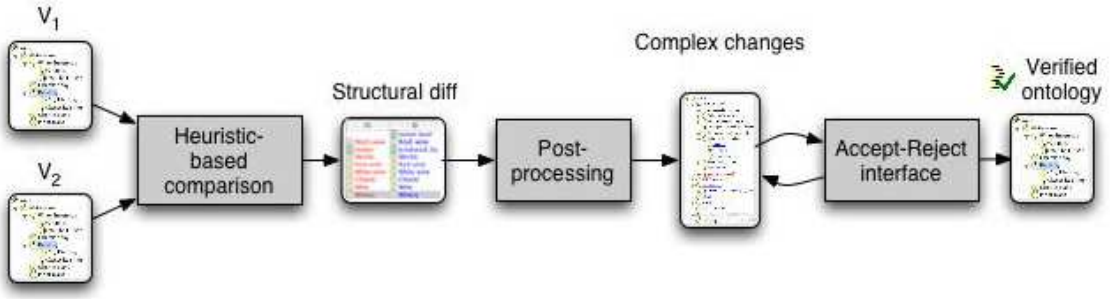
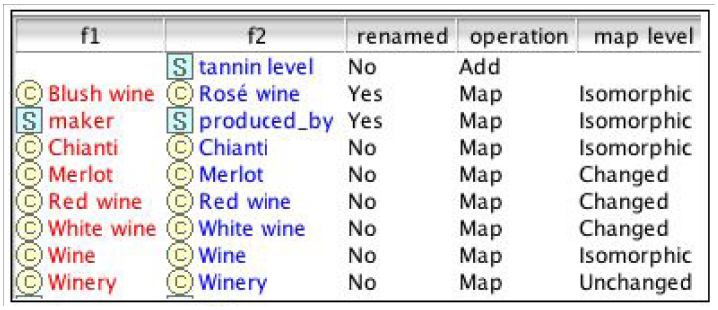


Figure 4.2: The architecture of the PromptDiff

Figure 4.2 shows the overall architecture of the PromptDiff ontology-versioning system. Two versions of an ontology, v_1 and v_2, are inputs to the system. The heuristic-based algorithm for comparing ontology versions analyzes the two versions and automatically produces a diff between v_1 and v_2 called a structural diff (Figure 4.3). The post-processing module uses the diff to identify complex changes. The results are presented to the user through the intuitive interface. The user then has the option of accepting or rejecting changes and these actions are reflected in the updated diff.



f1	f2	renamed	operation	map level
	tannin level	No	Add	
Blush wine	Rosé wine	Yes	Map	Isomorphic
maker	produced_by	Yes	Map	Isomorphic
Chianti	Chianti	No	Map	Isomorphic
Merlot	Merlot	No	Map	Changed
Red wine	Red wine	No	Map	Changed
White wine	White wine	No	Map	Changed
Wine	Wine	No	Map	Isomorphic
Winery	Winery	No	Map	Unchanged

Figure 4.3: The structural diff showing the difference between two versions

Given two versions of an ontology O, v_1 and v_2, a structural diff between v_1 and v_2, is a set of pairs $\langle r_1, r_2 \rangle$ where:

- $r_1 \in v_1$ or $r_1 = null$, $r_2 \in v_2$ or $r_2 = null$
- r_2 is an image of r_1 (matches r_1), that is, r_1 became r_2. If r_1 or r_2 is null, then we say that r_2 or r_1 respectively does not have a match.
- Each resource from v_1 and v_2 appears in at least one pair.
- For any resource r_1, if there is at least one pair containing r_1, where $r_2 \neq null$, then there is no pair containing r_1 where $r_2 = null$. The same is true for r_2.

The PromptDiff algorithm consists of two parts: (1) an extensible set of heuristic matchers and (2) a fixed-point algorithm to combine the results of the matchers to produce a structural diff between two versions. Each matcher employs a small number of structural properties of the ontologies to produce matches. The fixed-point step invokes the matchers repeatedly, feeding the results of one matcher into the others, until they produce no more

changes in the diff. Then the differences found by the algorithm are presented to the user who is responsible to accept or reject them.

4.3.1 Heuristic Matchers

The PromptDiff algorithm combines an arbitrary number of heuristic matchers, each of which looks for a particular property in the unmatched frames. The heuristic matchers compare two ontology versions looking for the following situations:

- **Resources of the same type with the same name.** In general, if $r_1 \in K_1$ and $r_2 \in K_2$ and r_1 and r_2 have the same name and type, then r_1 and r_2 match.

- **Single unmatched sibling.** In general, if $c_1 \in K_1$ and $c_2 \in K_2$, c_1 and c_2 match, and each of the classes has exactly one unmatched subclass, $subC_1$ and $subC_2$, respectively, then $subC_1$ and $subC_2$ match.

- **Siblings with the same suffixes or prefixes.** In general, if $c_1 \in K_1$ and $c_2 \in K_2$, c_1 and c_2 match, and the names of all subclasses of c_1 are the same as the names of all subclasses of c_2 except for a constant suffix or prefix, then the subclasses match.

- **Single unmatched Property.** In general, if $c_1 \in K_1$ and $c_2 \in K_2$, c_1 and c_2 match, and each of the classes has exactly one unmatched property, p_1 and p_2 respectively, and p_1 and p_2 have the same domain and range, then p_1 and p_2 match.

- **Unmatched inverse properties.** If a knowledge model allows definition of inverse relationships, then those relationships can be used to create matches as well. In general, if $p_1 \in K_1$ and $p_2 \in K_2$, p_1 and p_2 match, $invP_1$ and $invP_2$ are inverse properties for p_1 and p_2 respectively, and $invP_1$ and $invP_2$ are unmatched, then $invP_1$ and $invP_2$ match.

- **Split classes.** In general, if $c_0 \in K_1$ and $c_1 \in K_2$ and $c_2 \in K_2$, and for each instance of c_0, its image is an instance of either c_1 or c_2, then c_0 was split into c_1 and c_2. A similar matcher identifies classes that were merged.

4.4 Semversion

SemVersion [41] is a Java library for providing versioning facilities to RDF data. It is based on RDF/RDFS, so it can be used for any ontology language built or adapted to this data model.

Semversion offers an easy to use (and thus, integrate with) API that closely follows the usual functions and concepts of CVS [6]. To commit a new version, a user can either provide the complete contents of the version (which is an RDF model, i.e., simply a set of triples), or a *diff*, that is, the *change* that is to be applied on a preexisting version to create the new one.

At the implementation level (Figure 4.4), persistence is handled by RDF2Go [3], which provides common storage interfaces over triple- and quad-stores (SemVersion uses the abstraction of the latter), such as Jena [4], Sesame [5], YARS [6], NG4J [7], etc. SemVersion stores each version of an RDF model as a unique independent graph that contains the whole model.

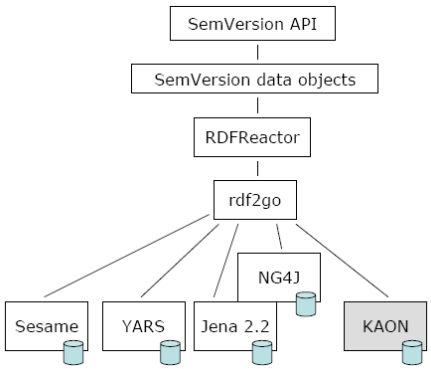


Figure 4.4: The Layered Architecture of SemVersion

Diffs serve two purposes: First, SemVersion allows to compute (structural and semantic) diffs between two arbitrary chosen models, to inform the user about changes. This allows collaborative ontology engineering. Second, diffs can be used in an update

[3] http://ontoware.org/projects/rdf2go/
[4] http://jena.sourceforge.net/
[5] http://www.openrdf.org/
[6] http://sw.deri.org/2004/06/yars/
[7] http://sites.wiwiss.fu-berlin.de/suhl/bizer/ng4j/

command to apply changes to a remotely stored model. When dealing with very large models, it might not be feasible (nor efficient) to transfer the complete model, if only a small fraction has changed.

Semversion provides three type of *diff*s analyzed at the next sections.

4.4.1 Set-based Diff

For versioning, the set-based diff is simply the set-theoretic difference of two RDF triple sets. Such diffs can be computed by simple set arithmetics for triple sets that contain only URIs and literals.

4.4.2 Structural Diff

Without the presence of blank nodes, the set-based diff is the same as the structural diff. With blank nodes, the set-based diff considers all blank nodes to be different and reports all statements involving blank nodes both as added and as removed.

SemVersion also handles the problem of uniquely identifying blank nodes. Blank nodes cannot be globally identified, as they lack a URI, and this poses a challenge at diff algorithms. This is overcome by adding a property to the blank nodes leading to a URI, effectively treating them, from that point on, as normal nodes. This procedure is called *blank node enrichment*. Other tools that process the RDF data are expected not to remove this property, so this will survive the roundtrip "extract a version from the repository, manipulate it in some ontology editor, reinsert the changes at the repository to create a new version", so that SemVersion can understand whether two blank nodes are the same. If this URI is missing, then SemVersion treats the node as new (since creating a new node from an external tool would be missing this, of course).

4.4.3 Semantic Diff

The semantic difference has to take the semantics of the used ontology language into account.

An intuitive way to understand the concept of a sematic diff goes like this: Let's

assume we use RDF Schema as our ontology language, and have two versions (K and K') of an RDFS ontology. Now, in order to compute the semantic RDFS diff, we use the closure of K ($C(K)$). Then we do the same for K' ($C(K')$). Now we calculate a structural (syntactical, set-based) diff on $C(K)$ and $C(K')$. This is not the same as the structural diff between K and K'. If the structural diff of two models is empty, then the semantic diff must also be empty. The inverse is not necessarily true: There might be two different RDF Knowledge Bases which encode the same semantic model, resulting in an empty semantic diff, but a nonempty structural diff.

4.5 RDF_utils

RDF_utils is developed as part of the KnoBot project. KnoBot is an RDF-based content management system which stores all its data in a Jena Model. It is designed to allow decentralized exchange of information founded on trust relationships between individual persons/agents. While it is designed to maximally comply with standards and best practices it offers novel features (e.g. relevance based aggregation).

RDF_utils is a utility tool for dealing with RDF data, it provides the following features:

- **Leanify**: Remove redundant statements (and anonymous nodes) from rdf-graphs
- **Diff**: Show the difference between two rdf-graphs

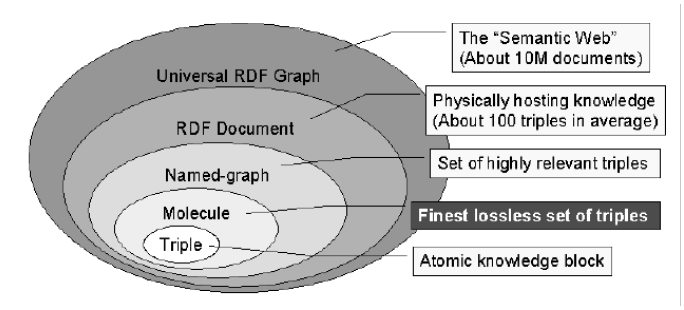


Figure 4.5: The granularity of the Semantic Web ranges from the universal graph to triple.

The main difference of this work to others is the choice of the level of granularity (Figure 4.5). The problem of granularity is well explained in [15]. RDF documents and

named graphs are too coarse for some particular application needs, such as in tracking provenance of an RDF graph. In this case, the overlap of the graph at hand with other graphs is a key to identify its provenance. But a named graph can't be used to express an overlap, as it will generally contain irrelevant triples too, unless explicitly calculating the intersection. On the other hand, triple-level is too fine-grained, due to the case of blank nodes. For example, see the RDF graphs of Figure 4.6. The first one shows an unnamed resource (blank node) with surname 'Ding' and first name 'Li'. The second graph is identical, while the third described another 'Ding' person, in particular 'Zhongli Ding'. If the triple-based overlap was meant to be used, the first and the third graph would appear that they share a common triple, while in fact the triples describe different people. This is due to the lack of universal identity of blank nodes; their identity is only derived by the named resources or literals connected to them. Clearly, when blank nodes are involved, equality of triples can't reliably be used as identification of equal RDF content.

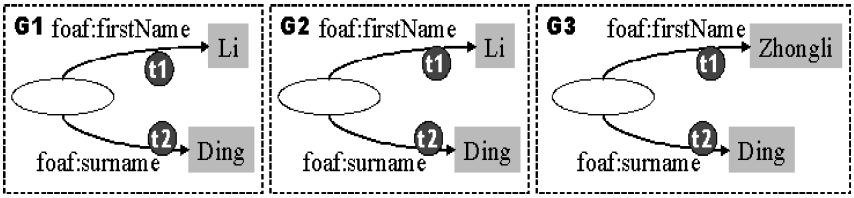


Figure 4.6: Three RDF graphs that show personal information from three sources. The first one asserts that a person who has first name 'Li' and surname 'Ding'.

In [16], the decomposition is defined as follows. An RDF graph decomposition consists of three elements (W, d, m): the background ontology W, the decompose operation $d(G, W)$ which breaks an RDF graph G into a set of sub-graphs $G* = G_1, G_2, ..., G_n$ using W, and the merge operation $m(G*, W)$ which combines all elements in $G*$ into the a unified RDF graph G' using W. In addition, a decomposition must be lossless such that $for\ any\ RDF graph\ G, G = m(d(G, W), W)$.

RDF molecules are defined as the finest and lossless subgraphs of a graph G according to a decomposition (W, d, m). Worth of note is that this concept is very similar to the notion of *Minimum Self-contained Graphs* (MSG), described in [40], one of the differences being that molecules also consider an arbitrary reasoning -the "background ontology"-

while MSG deals only with RDF).

4.6 CWM of w3c

CWM is part of SWAP, a Semantic Web Application Platform. SWAP consists of tools and applications to manipulate RDF graphs much like traditional tools manipulate text files. CWM is a command-line tool, written in python, for processing RDF in both the standard XML encoding and an experimental encoding, Notation3 [5].

CWM offers a utility that allows the user to compute the delta between two Knowledge Bases and then to apply the delta on the first Knowledge Base to obtain the second. CWM distinguishes two types of RDF deltas:

- **Weak delta**. Gives enough information to apply it to exactly the Knowledge Base it was computed from.

- **Strong delta**. Specifies the changes in a context independent manner. The difference is not in the format of the output but in the information a particular delta gives.

For example, if bank account numbers are globally unique, then a blank node that represents a bank account can be identified by a particular bank account. In OWL terms, if $bank : accountNumber$ is an $owl : InverseFunctionalProperty$, then the node must be the $owl : sameAs$ any other node with the same account number. In that case, the delta will be **strong**. If however, many accounts can have the same number, applying that delta to another knowledge base may inadvertently alter the wrong account. The delta is **weak**.

In order to produce strong deltas CWM uses the $owl : FunctionalProperty$ and $owl : InverseFunctionalProperty$ to assign labels to blank nodes in order to uniquely identify them. Strong deltas are provided if sufficient information can be found in the Web to fully label the input graphs.

4.6.1 Patch file format

By analogy to the text diff, there is a need not only for a difference-finding algorithm, but for a patch file format. Such a format needs:

- A way to uniquely identify what is changing.
- A way to distinguish between the pieces added and those subtracted.

It is straightforward to pinpoint the parts of the Knowledge Base that have changed when all nodes are named, but less so in the presence of anonymous nodes. To identify what is changing, Notation3 expressions are used and three new terms are introduced. For example:

```
 prefix diff:  < http://www.w3.org/2004/delta# >.
{ ?x bank:accountNo "1234578"; bank:balance 4000}
diff:replacement
{ ?x bank:accountNo "1234578"; bank:balance 3575}.
```

This one new property replacement can express any change. Deletions can be written {...} `diff:replacement` {} and additions can be written {} `diff:replacement` {...}.

The second alternative is very similar but involves two properties, one for inserting and one for deleting:

```
{ ?x bank:accountNo "1234578"}
    diff:deletion { ?x bank:balance 4000};
diff:insertion { ?x bank:balance 3575}.
```

The form using `diff:insertion` and `diff:deletion` is implemented in CWM.

4.7 Jena

Jena is a Semantic Web toolkit for Java programmers. The heart of the Semantic Web recommendations is the RDF Graph as a universal data structure. Jena similarly has the Graph as its core interface around which the other components are built.

Jena provides tools, including: a Java model/graph API, an RDF Parser (supporting an N-Triples filter), a query system based on RDQL , support classes for DAML+OIL ontologies and persistent/in-memory storage on BerkeleyDB or various other storage implementations. Due to its storage abstraction, Jena enables new storage subsystems to be integrated. To facilitate querying, Jena provides statement-centric methods for manipulating an RDF model as a set of RDF triples and resource-centric methods for manipulating an RDF model as a set of resources with properties, as well as built-in support for RDF containers. Jena contains Joseki RDF server, a server accepting SOAP and HTTP requests to query RDF resources. The latest version of Jena and Joseki support SPARQL.

Jena does not provide a mechanism to compare two RDF graphs and find the differences between them (i.e. compute the delta). But it provides a mechanism that decides whether two RDF graphs are isomorphic or not.

Jena follows the approach intoduced at [8] to decide isomorphism between graphs. A signature is created for each node (named or unnamed) assuming its position in the graph. Nodes that have the same signatures between the two graphs are matched. If all the nodes of the two graphs match then the graphs are isomorphic

4.8 pOWL

Powl, a web based ontology management tool. Its capabilities include parsing, storing, querying, manipulating, versioning, serving and serializing RDF and OWL knowledge bases for different target audiences.Powl is implemented in the web scripting language PHP.

Powls architecture consists of 4 stacked tiers, while trying to minimize dependencies and supplying clean interfaces between tiers. It consists of the following tiers:

- Powl store • SQL compatible relational database.
- RDFAPI, RDFSAPI, OWLAPI • layered APIs for handling RDF, RDFS and OWL.
- Powl API • containing classes and functions to build web applications on top of those APIs.

- User interface • a set of PHP pages combining widgets provided by Powl API for accessing (browsing, viewing, editing) model data in a Powl store.

To enable domain experts to collaboratively develop shared conceptualizations based on the Ontology Web Language a key requirement is to support a versioning strategy. In order to suport versioning, pOWL does provide a mechanism to compare Ontologies and find the differences between them, but supposes that all the changes were made through the pOWL platform and so they are tracked. One editing action by the user may be complex, but every editing action can be decomposed into smaller editing actions (Figure 4.7) and finally into adds and removes of RDF triples to or from the RDF model.

Powl enables rollback of every particular editing action by determining if the involved triples are still present (if added) or still missing (if removed). A parent action thus may only be rolled back if all sub-actions may be rolled back as well.

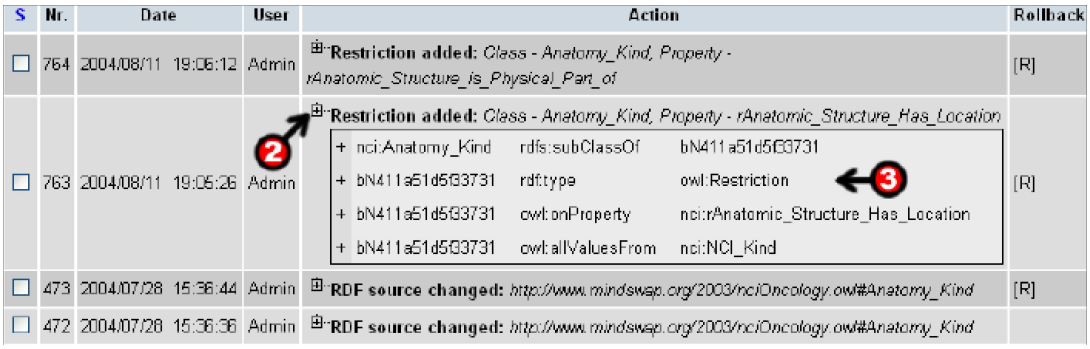


Figure 4.7: pOWL Versioning.

4.9 Summary Comparison

Table 4.9 presents a number of basic features which are then used for providing an overview of the functionality offered by each of the previously described systems.

The first five features correspond to the differential functions already introduced (i.e. Δ_e, Δ_c, Δ_d, Δ_{dc} and Δ_{ed}) and indicate if a system uses the comparison function or not. The next two features correspond to the two different change operations semantics introduced (i.e. $\mathcal{U}_p$ and $\mathcal{U}_{ir}$).

Features	Description
Δ_e	Use Δ_e as differential function
Δ_c	Use Δ_c as differential function
Δ_d	Use Δ_d as differential function
Δ_{dc}	Use Δ_{dc} as differential function
Δ_{ed}	Use Δ_{ed} as differential function
$\mathcal{U}_p$	Use $\mathcal{U}_p$-semantics
$\mathcal{U}_{ir}$	Use $\mathcal{U}_{ir}$-semantics
Heuristic matchers	Use heuristic matchers to detect changes
Isomorphism	Decide isomprphism between two RDF graphs
Change Log	Maintain the sequence of applied changes

Table 4.1: Features of the comparison

Heuristic matchers are a kind of rules that are used in order to decide if something has changes at a Knowledge Base. Unfortunately, the heuristic matchers are language specific i.e. depend on the semantics of the underlying language. For example the heuristic matchers that need to be used for RDF differ from those used for OWL.

The *Isomorphism* is the ability to decide whether two RDF graphs, which may contain blank nodes, have the same structure. The tools that implement this feature may not produce a delta as a result, but just decide isomorphism between the graphs.

The *Change Log* is a registry that records the actions that occurred in the versioning system, for example the steps taken to create a new version from an older one. If the actions are completely recorded, one could traverse this log and apply the actions in the order that they occurred and reach the same result. The change log is also useful for a user that wants to understand the changes made by someone else, to see the way they work and possibly spot errors.

Features	Systems							
	OntoView	**PromptDiff**	**SemVersion**	**RDF_Utils**	**CWM**	**Jena**	**Powl**	**SWKM**
Δ_e	no	no	yes	yes	yes	no	no	yes
Δ_c	no	no	yes	no	no	no	no	yes
Δ_d	no	no	no	no	no	no	no	yes
Δ_{dc}	no	no	no	no	no	no	no	yes
Δ_{ed}	no	no	no	no	no	no	no	yes
$\mathcal{U}_p$	no	no	yes	yes	yes	no	no	yes
$\mathcal{U}_{ir}$	no	no	no	no	no	no	no	yes
Heuristic matchers	yes	yes	no	no	no	no	no	no
Isomorphism	no	no	no	yes	yes	yes	no	not yet
Change Log	no	no	no	no	no	no	yes	no

Table 4.2: Features comparison of versioning systems and tools

The differential function Δ_e is utilized by SemVersion, RDF_Utils, CWM and SWKM, while Δ_c is used by SemVersion and SWKM. The other differential functions (i.e. Δ_d,

Δ_{dc} and Δ_{ed}) are used only by SWKM. Considering the change operation semantic, $\mathcal{U}_p$ is used by SemVersion, RDF_Utils, CWM and SWKM, while $\mathcal{U}_{ir}$ is used only by SWKM. PromptDiff and OntoView detect changes using heuristic matches. RDF_Utils, CWM and Jena can detect if two RDF graphs are isomorphic. Finally, Powl uses a change log to keep track of the changes in order not to have to compute the delta.

Chapter 5

Conclusion and Future Work

One approach for computing the difference between two RDF KBs is to take the difference between the sets of triples forming the two KBs (along with some refinements such as taking into account blank nodes). Another approach (useful for versioning) is to identify a set of change operations that will transform one KB into another. In this thesis we investigated the latter approach and studied different semantics for this computation as well as properties like minimality, correctness, semantic identity, non-redundancy, reversibility and composability of the produced Deltas. Most of the existing RDF differential tools [4, 41, 15] rely on the $(\Delta_e, \mathcal{U}_p)$ pair. Semversion [41] offers also $(\Delta_c, \mathcal{U}_p)$ for the case where the K is complete (we have proved that in such cases this approach yields correct results). None of the works (theoretical or practical) has used Δ_d, Δ_{dc} or Δ_{ed}.

Recall that we have shown that $(\Delta_{dc}, \mathcal{U}_{ir})$ is better than $(\Delta_c, \mathcal{U}_p)$ not only because $(\Delta_{dc}, \mathcal{U}_{ir})$ does not require the KBs to be complete, but also because it returns smaller in size Deltas. Moreover, we have shown that $(\Delta_e, \mathcal{U}_p)$ and $(\Delta_c, \mathcal{U}_{ir})$ satisfy properties like reversibility and composability, while $(\Delta_{dc}, \mathcal{U}_{ir})$ and $(\Delta_{ed}, \mathcal{U}_p)$ do not. On the other hand, $(\Delta_e, \mathcal{U}_p)$ and $(\Delta_{ed}, \mathcal{U}_p)$ do not satisfy semantic identity. We have also identified the cases where $(\Delta_d, \mathcal{U}_{ir})$ is beneficial (recall that Δ_d gives the minimum in size Deltas).

Concerning the size criterion, Δ_d produces the smallest in size result in the general case, while Δ_{ed} produces smaller in size results than Δ_e, and Δ_{dc} smaller in size results than Δ_c. From the experiments we have also seen that at most cases Δ_e produces smaller in size results than Δ_{dc} (i.e. the ordering is $|\Delta_d| \leq |\Delta_{ed}| \leq |\Delta_e| \leq |\Delta_{dc}| \leq |\Delta_c|$) However,

if $K \sim K'$ then Δ_e and Δ_{ed} may produce a very big in size result while Δ_{dc} and Δ_c produce an empty result.

The pairs that are always correct are: $(\Delta_e, \mathcal{U}_p)$, $(\Delta_{ed}, \mathcal{U}_p)$, $(\Delta_{dc}, \mathcal{U}_{ir})$ and $(\Delta_c, \mathcal{U}_{ir})$. Each of them is beneficial under different scenarios:

(i) If we need a pair that satisfies semantic identity, reversibility and composability then the pair $(\Delta_c, \mathcal{U}_{ir})$ is the most appropriate.

(ii) If we need a pair that satisfies reversibility and composability, but does not satisfy semantic identity then the pair $(\Delta_e, \mathcal{U}_p)$ is the most appropriate.

(iii) If we need a pair that satisfies semantic identity, but does not satisfy reversibility and composability then the pair $(\Delta_{dc}, \mathcal{U}_{ir})$ is the most appropriate.

(iv) If we need a pair that is correct but does not satisfy semantic identity reversibility and composability then the pair $(\Delta_{ed}, \mathcal{U}_p)$ is the most appropriate.

The most common of the above scenarios (as described at Section 1.3.1) are (i) and (ii) where we need pair of differential function and change operation semantics that satisfies reversibility and composability. So the pairs $(\Delta_c, \mathcal{U}_{ir})$ and $(\Delta_e, \mathcal{U}_p)$ are beneficial at the most common scenarios.

Note also that the complexity of Δ_e ($O(N)$) and Δ_{ed} ($O(N\gamma)$) is smaller than the complexity of Δ_c ($O(N^2)$) and Δ_{dc} ($O(M\gamma)$). This can be seen also from the experimental results where the time for Δ_{dc} and Δ_c is greater than the time for Δ_e. Moreover, for all differential functions that are correct using $\mathcal{U}_{ir}$ semantics (i.e. Δ_c and Δ_{dc}) the execution cost is higher because $\mathcal{U}_{ir}$ semantics require the computation of the closure and the reduction. Concerning the execution of $\mathcal{U}_{ir}$ operations, related algorithms include [35], while a similar in spirit approach for RDF has already been implemented for the RUL language [28].

We have exploited the properties of the various differential functions presented in this thesis for building versioning services on top of SW repositories [36]. Specifically, we have implemented all the reported differential functions and both $\mathcal{U}_p$ and $\mathcal{U}_{ir}$ change operation semantics.

In comparison with belief contraction-revision (e.g. [20, 26, 17]), these theories consider KBs as logic theories and focus on what the result of applying a contraction/ revision operation on a KB should be. In our setting, the destination KB is known, i.e. it is K', so the focus is given on the transition from K to K'[1].

We plan to extend the current work, theoretical and practical, in order to handle blank nodes efficiently. In particular, we plan to adopt a technic that allows us to decide if two RDF graphs that include blank nodes, are isomorphic or not. Moreover, this technic will be able to match the blank nodes of the two graphs in such way that a minimum in size delta is produced. The blank node matching step will ne a preprocessing step; therefore, it will not affect the current implementation of the differential functions.

Finally, in the case of very large KBs that do not fit in memory it would be possible to have a diff implementation over RQL.

[1]Note that $\mathcal{U}_p$ and $\mathcal{U}_{ir}$ are not proposed as general purpose change operations, but only for executing the results of the differential functions we have defined in this thesis.

Bibliography

[1] S. Agarwal, D. Starobinski, and A. Trachtenberg. "On the Scalability of Data Synchronization Protocols for PDAs and Mobile Devices". In *IEEE Network (Special Issue on Scalability in Communication Networks), Vol. 16, No.4, pp.2228*, 2002.

[2] C. E. Alchourrón, P. Gärdenfors, and D. Makinson. "On the Logic of Theory Change: Partial Meet Contraction and Revision Functions". *J. Symb. Log.*, 50(2):510–530, 1985.

[3] S. Auer. "Powl - A Web Based Platform for Collaborative Semantic Web Development". In *Proc. of 1st Workshop Workshop Scripting for the Semantic Web (SFSW'05)*, Hersonissos, Greece, May 2005.

[4] T. Beners-Lee and D. Connoly. "Delta: An Ontology for the Distribution of Differences Between RDF Graphs", 2004. http://www.w3.org/DesignIssues/Diff (version: 2006-05-12).

[5] T. Beners-Lee, D. Connoly, and S. Hawke. Semantic web tutorial using n3. In *Twelfth International World Wide Web Conference*, Budapest, Hungary, May 2003.

[6] B. Berliner. "CVS II: Parallelizing Software Development". In *Procs of the USENIX Winter 1990 Technical Conference*, pages 341–352, Berkeley, CA, 1990.

[7] D. Brickley and R. V. Guha. "RDF Vocabulary Description Language 1.0: RDF Schema, W3C Recommendation", February 2004. http://www.w3.org/TR/rdf-schema/.

[8] J. J. Carroll. "Matching RDF graphs". In *Procs of the ISWC'02*, pages 5–15, Italy, Oct. 2002.

[9] S. Chien, V. Tsotras, and C. Zaniolo. Eficient Management of Multiversion Documents by Object Referencing. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 291–300, Roma, Italy, 2007.

[10] V. Christophides, D. Plexousakis, M. Scholl, and S. Tourtounis. "On Labeling Schemes for the Semantic Web". In *Procs. of WWW'03*, pages 544–555, Budapest, Hungary, May 2003.

[11] R. Cloran and B. Irwin. "Transmitting RDF graph deltas for a Cheaper Semantic Web". In *Procs. of SATNAC'2005*, South Africa, September 2005.

[12] G. Cobena, S. Abiteboul, and A. Marian. Detecting Changes in XML Documents. In *Proceedings of the International Conference on Data Engineering (ICDE)*, Heidelberg, Germany, 2001.

[13] R. Conradi and B. Westfechtel. "Version models for software configuration management". *ACM Comput. Surv.*, 30(2):232–282, 1998.

[14] P. Dadam, V. Lum, and H.-D. Werner. Integration of Time Versions into a Relational Database System. In *VLDB '84: Proceedings of the 10th International Conference on Very Large Data Bases*, pages 509–522, San Francisco, CA, USA, 1984.

[15] L. Ding, T. Finin, A. Joshi, Y. Peng, P. da Silva, and D. McGuinness. "Tracking RDF Graph Provenance using RDF Molecules". In *Procs of ISWC'05*, Galway,Ireland, November 2005.

[16] Li Ding, Tim Finin, Yun Peng, Paulo Pinheiro da Silva, and Deborah L. McGuinness. Tracking RDF Graph Provenance using RDF Molecules. Technical report, UMBC, April 2005.

[17] G. Flouris. *"On Belief Change and Ontology Evolution"*. PhD thesis, Computer Science Department, University of Crete, Greece, 2006.

[18] FORTH-ICS. "The ICS-FORTH RDFSuite: High-level Scalable Tools for the Semantic Web", 2005. http://139.91.183.30:9090/RDF/.

[19] J. Petersson F. Piazza P. Puliti G. Tummarello, C. Morbidoni. "RDFGrowth, a P2P annotation exchange algorithm for scalable Semantic Web applications". In *1st Annual International Conference on Mobile and Ubiquitous Systems MobiQuitous*, Boston, MA, August 2004.

[20] P. Gärdenfors. "Belief Revision: An Introduction". In *Belief Revision*, pages 1–20. Cambridge University Press, 1992.

[21] C. Gutierrez, C. Hurtado, and A. Mendelzon. "Foundations of Semantic Web Databases". In *In 23 ACM Symposium on Principles of Database Systems (PODS)*, 2004.

[22] P. Hayes. "RDF Semantics, W3C Recommendation", February 2004. http://www.w3.org/TR/rdf-mt/.

[23] J. Heflin, J. Hendler, and S. Luke. "Coping with Changing Ontologies in a Distributed Environment". In *Procs of AAAI-99 Workshop on Ontology Management*, Florida, July 1999.

[24] M. Klein, D. Fensel, A. Kiryakov, and D. Ognyanov. "Ontology versioning and change detection on the web". In *Procs of EKAW'02*, pages 197–212, Siguenza, Spain, Oct 2002.

[25] M. Klein and N. Noy. "A component-based framework for ontology evolution". In *InWorkshop on Ontologies and Distributed Systems at IJCAI-03*, Acapulco, Mexico, 2003.

[26] S. Konieczny and R. P. Perez. "Propositional Belief Base Merging or How to Merge Beliefs/Goals Coming from Several Sources and Some Links With Social Choice Theory". *European Journal of Operational Research*, 160(3):785–802, 2005.

[27] George Konstantinidis, Giorgos Flouris, Grigoris Antoniou, and Vassilis Christophides. Ontology evolution: A framework and its application to rdf. 2007. In

Proceedings of the Joint ODBIS and SWDB Workshop on Semantic Web, Ontologies, Databases (SWDB-ODBIS-07).

[28] M. Magiridou, S. Sahtouris, V. Christophides, and M. Koubarakis. "RUL:A Declarative Update Language for RDF". In *Procs of ISWC'05*, pages 506–521, Galway, Ireland, Nov 05.

[29] A. Marian, S. Abiteboul, G. Cobena, and L. Mignet. Change-Centric Management of Versions in an XMLWarehouse. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 581–590, Roma, Italy, 2001.

[30] Y. Minsky, A. Trachtenberg, and R. Zippel. Set reconciliation with nearly optimal communication complexity. In *International Symposium on Information Theory*, page 232, 2001.

[31] M. Klein N. F. Noy, S. Kunnatur and M. A. Musen. "Tracking Changes During Ontology Evolution". In *Procs of ISWC'04*, pages 259–273, Hisroshima, Japan, November 2004.

[32] N. F. Noy and M. A. Musen. "PromptDiff: A Fixed-point Algorithm for Comparing Ontology Versions". In *Procs of AAAI-02*, pages 744–750, Edmonton, Alberta, July 2002.

[33] N. F. Noy and M. A. Musen. "Ontology versioning in an ontology management framework". *IEEE Intelligent Systems*, 19(4):6–13, 2004.

[34] P. Plessers and O. De Troyer. "Ontology Change Detection Using a Version Log". In *Procs of ISWC'05*, pages 578–592, Galway,Ireland, November 2005.

[35] J. A. La Poutre' and J. van Leeuwen. "Maintenance of Transitive Closures and Transitive Reductions of Graphs". In *Procs of the Intern. Workshop on Graph-Theoretic Concepts in Computer Science*, pages 106–120, July 1987.

[36] Y. Theoharis, V. Christophides, and G.Karvounarakis. "Benchmarking Database Representations of RDF/S Stores". In *Procs of ISWC'05*, pages 685–701, Galway, Ireland, Nov 05.

[37] Y. Theoharis, G. Georgakopoulos, and V. Christophides. On the Synthetic Generation of Semantic Web Schemas. In *Procs of the Joint ODBIS & SWDB Workshop on Semantic Web, Ontologies, Databases. Colocated with VLDB2007*, September 2007.

[38] W. F. Tichy. "RCS-•a system for version control". *Software Practice & Experience*, 15(7):637–654, July 1985.

[39] A. Tridgell. "Efficient algorithms for sorting and synchronization". Phd thesis, The Australian National University, February 1999.

[40] Giovanni Tummarello, Christian Morbidoni, Paolo Puliti, and Francesco Piazza. Signing individual fragments of an rdf graph. In *Proceedings of the World Wide Web conference*, 2005.

[41] M. Volkel, W. Winkler, Y. Sure, S. Ryszard Kruk, and M. Synak. "SemVersion: A Versioning System for RDF and Ontologies". In *Procs of ESWC'05.*, Heraklion, Crete, May 2005.

[42] Z. Zhang, L. Zhang, C. Lin, Y. Zhao, and Y. Yu. "Data Migration for Ontology Evolution". In *Poster Proceedings ISWC'03*, Sanibel Island, Florida, USA, October 2003.

Chapter 6

Appendix: Proofs

6.1 Delta Size Proofs

LEMMA. 1 For any pair of knowledge bases K and K' it holds:

$$\begin{aligned} |\Delta_d| &\leq |\Delta_{ed}| \leq |\Delta_e| \\ |\Delta_d| &\leq |\Delta_{dc}| \leq |\Delta_c| \end{aligned}$$

Proof:

First we list a number of properties that are used for the proof

(1) $K \subseteq C(K) \Rightarrow K' - C(K) \subseteq K' - K.$

(2) $K' \subseteq C(K') \Rightarrow K - C(K') \subseteq K - K'.$

(3) $K' \subseteq C(K') \Rightarrow K' - C(K) \subseteq C(K') - C(K).$

(4) $K \subseteq C(K) \Rightarrow K - C(K') \subseteq C(K) - C(K')$

The formula for deletion is the same for both Δ_d and Δ_{ed}, so if we consider additions then from (1) we get $|\Delta_d| \leq |\Delta_{ed}|$. Furthermore, from (1) and (2) it follows that $|\Delta_d| \leq |\Delta_e|$. The formula for additions is the same for Δ_{ed} and Δ_e, so if we consider deletions, from (2) we get $|\Delta_{ed}| \leq |\Delta_e|$. The formula for additions is the same for both Δ_d and Δ_{dc}, so if we consider deletions from (4) we get $|\Delta_d| \leq |\Delta_{dc}|$. From (3) and (4) it follows that $|\Delta_d| \leq |\Delta_c|$. Finally,

the formula for deletions is the same for both Δ_c and Δ_{dc}. If we consider additions then from (3) we get $|\Delta_{dc}| \leq |\Delta_c|$. $\diamond$

Prop. 1 If $K \subseteq K'$ then the following relationships also hold:

$$\begin{aligned} |\Delta_e| &= |\Delta_{ed}| \\ |\Delta_d| &= |\Delta_{dc}| \end{aligned}$$

Proof:

From the definition of closure we know that $K \subseteq K' \Rightarrow C(K) \subseteq C(K')$. From the above it follows that $K - K' = C(K) - C(K') = K - C(K') = \emptyset$ i.e. the result of the five differential functions does not contain deletions of triples. Δ_e and Δ_{ed} have the same formula for additions so $|\Delta_e| = |\Delta_{ed}|$ the same holds for Δ_d and Δ_{dc} i.e. $|\Delta_d| = |\Delta_{dc}|$. $\diamond$

Prop. 2 If $K \supseteq K'$ then the following relationships also hold:

$$\begin{aligned} |\Delta_d| &= |\Delta_{ed}| \\ |\Delta_c| &= |\Delta_{dc}| \end{aligned}$$

Proof:

From the definition of closure we know that $K \supseteq K' \Rightarrow C(K) \supseteq C(K')$. From the above it follows that $K' - K = C(K') - C(K) = K' - C(K) = \emptyset$ i.e. the result of the five differential functions does not contain addition of triples. Δ_d and Δ_{ed} have the same formula for deletion so $|\Delta_d| = |\Delta_{ed}|$, the same holds for Δ_c and Δ_{dc}, i.e. $|\Delta_c| = |\Delta_{dc}|$. $\diamond$

Prop. 3 If $K = C(K)$ then the following relationships also hold:

$$|\Delta_d| = |\Delta_{dc}| = |\Delta_{ed}|$$

Proof:

If $K = C(K)$ then the formulas for Δ_d, Δ_{dc} and Δ_{ed} are identical and equal to $\{Add(t) \mid t \in K' - C(K)\} \cup \{Del(t) \mid t \in C(K) - C(K')\}$. This means that the those three differential functions also produce same in size results. $\diamond$

Prop. 4 If $K' = C(K')$ then the following relationships also hold:

$$\begin{aligned} |\Delta_c| &= |\Delta_{dc}| \\ |\Delta_e| &= |\Delta_{ed}| \end{aligned}$$

Proof:

If $K' = C(K')$ then the formulas for Δ_c and Δ_{dc} are identical and equal to $\{Add(t) \mid t \in C(K') - C(K)\} \cup \{Del(t) \mid t \in C(K) - C(K')\}$. Furthermore, the formulas for Δ_e and Δ_{ed} are identical and equal to $\{Add(t) \mid t \in C(K)' - K\} \cup \{Del(t) \mid t \in K - C(K')\}$. This means that Δ_c and Δ_{dc} produce same in size results. The same happens for Δ_e and Δ_{ed}. $\diamond$

Prop. 5 If $K = C(K')$ then the following relationships also hold:

$$\begin{aligned} |\Delta_d| = |\Delta_{dc}| &= |\Delta_c| = |\Delta_{ed}| = 0 \\ |\Delta_e| &\geq 0 \end{aligned}$$

Proof:

If $K = C(K')$ then:

(1) $K' - K = K' - C(K') = \emptyset$.

(2) $C(K') - C(K) = C(K') - C(C(K')) = C(K') - C(K') = \emptyset$.

(3) $K' - C(K) = K' - C(C(K')) = K' - C(K') = \emptyset$.

(4) $C(K) - C(K') = C(C(K')) - C(K') = C(K') - C(K') = \emptyset$

(5) $K - C(K') = C(K') - C(K') = \emptyset$

From (2) and (4) it follows that $\Delta_c = \emptyset \Leftrightarrow |\Delta_c| = 0$, while from (3) and (5) it follows that $\Delta_d = \emptyset \Leftrightarrow |\Delta_d| = 0$. Moreover, from (3) and (4) it follows that $\Delta_{dc} = \emptyset \Leftrightarrow |\Delta_{dc}| = 0$. From (1) and (5) it follows that $\Delta_{ed} = \emptyset \Leftrightarrow |\Delta_{ed}| = 0$. Finally, from (1) it follows that $\Delta_e^+ = \emptyset$, but $\Delta_e^- \supseteq \emptyset \Rightarrow |\Delta_e| \geq 0$. $\diamond$

Prop. 6 If $K' = C(K)$ then the following relationships also hold:

$$\begin{aligned} |\Delta_d| = |\Delta_{dc}| &= |\Delta_c| = 0 \\ |\Delta_e| = |\Delta_{ed}| &\geq 0 \end{aligned}$$

Proof:

If $K' = C(K)$ then:

(1) $K - K' = K - C(K) = \emptyset$.

(2) $C(K') - C(K) = C(C(K)) - C(K) = C(K) - C(K) = \emptyset$.

(3) $K' - C(K) = C(K) - C(K) = \emptyset$.

(4) $C(K) - C(K') = C(K) - C(C(K)) = C(K) - C(K) = \emptyset$

(5) $K - C(K') = K - C(C(K)) = K - C(K) = \emptyset$

From (2) and (4) it follows that $\Delta_c = \emptyset \Leftrightarrow |\Delta_c| = 0$, while from (3) and (5) it follows that $\Delta_d = \emptyset \Leftrightarrow |\Delta_d| = 0$. Moreover, from (3) and (4) it follows that $\Delta_{dc} = \emptyset \Leftrightarrow |\Delta_{dc}| = 0$. From (1) it follows that $\Delta_e^- = \emptyset$, while from (5) it follows that $\Delta_{ed}^- = \emptyset$. But $\Delta_e^+ = \Delta_{ed}^+ \supseteq \emptyset \Rightarrow |\Delta_e| = |\Delta_{ed}| \geq 0$. $\diamond$

Prop. 7 If $K \sim K'$ then the following relationships also hold:

$$\begin{aligned} |\Delta_d| = |\Delta_{dc}| &= |\Delta_c| = 0 \\ 0 \leq |\Delta_{ed}| &\leq |\Delta_{ed}| \end{aligned}$$

Proof:

Lemma 1 and Proposition 8 (found at section 6.4) prove the correctness of this proposition. $\diamond$

Prop. 12 For any pair of knowledge bases K and K' it holds:

$$\begin{aligned} |\Delta_e(K \to K')| &\leq |\Delta_{ed}(K \to K')| + |\Delta_{ed}(K' \to K)| \\ |\Delta_c(K \to K')| &\leq |\Delta_{dc}(K \to K')| + |\Delta_{dc}(K' \to K)| \end{aligned}$$

Proof:

$|\Delta_e(K \to K')| \leq |\Delta_{ed}(K \to K')| + |\Delta_{ed}(K' \to K)| \Leftrightarrow |K' - K| + |K - K'| \leq |K' - K| + |K - C(K')| + |K - K'| + |K' - C(K)| \Leftrightarrow 0 \leq |K - C(K')| + |K' - C(K)|$ That is true.

$|\Delta_c(K \to K')| \leq |\Delta_{dc}(K \to K')| + |\Delta_{dc}(K' \to K)| \Leftrightarrow |C(K') - C(K)| + |C(K) - C(K')| \leq |K' - C(K)| + |C(K) - C(K')| + |K - C(K')| + |C(K') - C(K)| \Leftrightarrow 0 \leq |K' - C(K)| + |K - C(K')|$ That is true. $\diamond$

6.2 Auxiliary Proofs

In this section we list a number of properties (the proofs of some of these are trivial and are therefore omitted). Later on we exploit these properties for proving a number of propositions and theorems.

(a) $R(R(K)) = R(K)$

(b) $C(R(K)) = C(K)$

(c) $A - (B - D) = (A - B) \cup (A \cap D)$

(d) If $A \subseteq D$ $(A - B) \cup (B \cap D) = A \cup (B \cap D)$

(e) $(B - A) \cap D = (B \cap D) - A$

(f) $(A \cup B) - D = (A - D) \cup (B - D)$

(g) $C - (A \cup B) = (C - A) \cap (C - B)$

Proof of (c)

$x \in A - (B - D) \Leftrightarrow$

$x \in A$ and $x \notin (B - D) \Leftrightarrow$

$x \in A$ and not $(x \in B$ and $x \notin D) \Leftrightarrow$

$x \in A$ and $(x \notin B$ or $x \in D)$

$(x \in A$ and $x \notin B)$ or $(x \in A$ and $x \in D) \Leftrightarrow$

$x \in (A - B)$ or $x \in (A \cap D) \Leftrightarrow$

$x \in (A - B) \cup (A \cap D)$ $\diamond$

Proof of (d)

$A \subseteq D \Rightarrow A \cap B \subseteq B \cap D$ (F)

$(A - B) \cup (B \cap D) = (A - (A \cap B)) \cup (B \cap D) \stackrel{(F)}{=} A \cup (B \cap D)$ $\diamond$

6.3 Correctness Proofs

Prop. 15 $\Delta_e(K \rightarrow K')^{\mathcal{U}_p}(K) \sim K'$

Proof:

$\Delta_e(K \rightarrow K')^{\mathcal{U}_p}(K) = (K - (K - K')) \cup (K' - K) \stackrel{(c)}{=} ((K' \cap K) \cup (K - K)) \cup (K' - K) =$

$(K' \cap K) \cup (K' - K) \stackrel{(c)}{=} K' - (K - K) = K'.\diamond$

Prop. 16 $\Delta_{ed}(K \to K')^{\mathcal{U}_p}(K) \sim K'$

Proof:
$\Delta_{ed}(K \to K')^{\mathcal{U}_p}(K) = (K - (K - C(K'))) \cup (K' - K) \stackrel{(c)}{=} ((K - K) \cup (K \cap C(K')) \cup (K' - K)) =$
$(K \cap C(K')) \cup (K' - K) \stackrel{(d)}{=} K' \cup (K \cap C(K'))$

Let $\mathtt{L}$=$K' \cup (K \cap C(K'))$ then

$K \cap C(K') \subseteq C(K') \Leftrightarrow K' \cup (K \cap C(K')) \subseteq K' \cup C(K') \Leftrightarrow K' \cup (K \cap C(K')) \subseteq C(K') \Leftrightarrow \mathtt{L} \subseteq C(K')$

$\emptyset \subseteq K \cap C(K') \Leftrightarrow K' \subseteq K' \cup (K \cap C(K')) \Leftrightarrow K' \subseteq \mathtt{L}$

From the above two inequations we get : $K' \subseteq \mathtt{L} \subseteq C(K') \Leftrightarrow \mathtt{L} \sim K'$. This means that $(\Delta_{ed}, \mathcal{U}_p)$ is correct.$\diamond$

Prop. 17 $\Delta_c(K \to K')^{\mathcal{U}_{ir}}(K) \sim K'$

Proof:
$\Delta_c(K \to K')^{\mathcal{U}_{ir}}(K) = R((C(K) - (C(K) - C(K'))) \cup (C(K') - C(K))) \stackrel{(c)}{=}$
$R(((C(K) \cap C(K')) \cup (C(K) - C(K))) \cup (C(K') - C(K))) = R((C(K) \cap C(K')) \cup (C(K') -$
$C(K))) \stackrel{(c)}{=} R(C(K') - (C(K) - C(K))) = R(C(K')) \sim K' \diamond$

Prop. 18 $\Delta_{dc}(K \to K')^{\mathcal{U}_{ir}}(K) \sim K'$

Proof:
$\Delta_{dc}(K \to K')^{\mathcal{U}_{ir}}(K) = R((C(K) - (C(K) - C(K'))) \cup (K' - C(K))) \stackrel{(c)}{=} R(((C(K) \cap C(K')) \cup$
$(C(K) - C(K))) \cup (K' - C(K))) = R((C(K) \cap C(K')) \cup (K' - C(K))) \stackrel{(d)}{=} R((C(K) \cap C(K')) \cup K')$

Let $\mathtt{Z}$=$R((C(K) \cap C(K')) \cup K')$ then

$C(K) \cap C(K') \subseteq C(K') \Leftrightarrow (C(K) \cap C(K')) \cap K' \subseteq C(K') \cap K' \Leftrightarrow R((C(K) \cap C(K')) \cap K') \subseteq$
$R(C(K') \cap K') \Leftrightarrow R((C(K) \cap C(K')) \cap K') \subseteq R(K') \Leftrightarrow \mathtt{Z} \subseteq R(K')$

$K' \subseteq (C(K) \cap C(K')) \cup K' \Leftrightarrow R(K') \subseteq R((C(K) \cap C(K')) \cup K') \Leftrightarrow R(K') \subseteq \mathtt{Z}$

From the above two inequalities it follows that: $R(K') \subseteq \mathtt{Z} \subseteq R(K') \Leftrightarrow \mathtt{Z} = R(K') \Leftrightarrow \mathtt{Z} \sim K'$

This means that $(\Delta_{dc}, \mathcal{U}_{ir})$ is correct.$\diamond$

Theorem 1 For any pair of valid knowledge bases K and K' it holds:

$$\Delta_e(K \to K')^{\mathcal{U}_p}(K) \sim \Delta_{ed}(K \to K')^{\mathcal{U}_p}(K) \sim$$
$$\Delta_c(K \to K')^{\mathcal{U}_{ir}}(K) \sim \Delta_{dc}(K \to K')^{\mathcal{U}_{ir}}(K) \sim K'$$

Proof:

The Propositions 15, 16, 17 and 18 prove the correctness of Theorem 1. $\diamond$

Theorem 2 $\Delta_d(K \to K')^{\mathcal{U}_{ir}}(K) \sim K'$ iff K is complete, or $C(K) - K \subseteq C(K')$.

Proof:

$\Delta_d(K \to K')^{\mathcal{U}_{ir}}(K) = R((C(K) - (K - C(K'))) \cup (K' - C(K))) \overset{(c)}{=} R(((C(K) \cap C(K')) \cup (C(K) - K)) \cup (K' - C(K))) = R((C(K) \cap C(K')) \cup (K' - C(K))) \cup (C(K) - K)) \overset{(d)}{=} R(((C(K) \cap C(K')) \cup K') \cup (C(K) - K))$

Let $\mathtt{W}$=$R(((C(K) \cap C(K')) \cup K') \cup (C(K) - K))$

$C(K) \cap C(K') \subseteq C(K') \Leftrightarrow ((C(K) \cap C(K')) \cup K') \cup (C(K) - K) \subseteq (C(K') \cup K') \cup (C(K) - K) \Leftrightarrow R(((C(K) \cap C(K')) \cup K') \cup (C(K) - K)) \subseteq R((C(K') \cup K') \cup (C(K) - K)) \Leftrightarrow \mathtt{W} \subseteq R(C(K') \cup (C(K) - K))$

$K' \subseteq (C(K) \cap C(K')) \cup K' \Leftrightarrow K' \cup (C(K) - K) \subseteq ((C(K) \cap C(K')) \cup K') \cup (C(K) - K) R(K' \cup (C(K) - K)) \subseteq R((C(K) \cap C(K')) \cup K') \cup (C(K) - K)) \Leftrightarrow R(K' \cup (C(K) - K)) \subseteq \mathtt{W}$

From the above two equivalances it follows that: $R(K' \cup (C(K) - K) \subseteq \mathtt{W} \subseteq R(C(K') \cup (C(K) - K))$

In order to prove that $W \sim K'$ we have to prove that $R(K') \subseteq W \subseteq R(C(K'))$, this is true at the above cases:

(a) $C(K) - K = \emptyset \Rightarrow C(K) = K$ i.e when K is complete

(b) if every triple $t \in C(K) - K$ can be inferred from K' i.e. $C(K) - K \subseteq C(K')$

$\diamond$

Theorem 3 $\Delta_c(K \to K')^{\mathcal{U}_p}(K) \sim K'$ iff K is complete.

Proof:

$\Delta_c(K \to K')^{\mathcal{U}_p}(K) = (K - (C(K) - C(K'))) \cup (C(K') - C(K)) \stackrel{(c)}{=} ((K \cap C(K')) \cup (K - C(K))) \cup (C(K') - C(K)) = (K \cap C(K')) \cup (C(K') - C(K)) \stackrel{(c)}{=} C(K') - (C(K) - K)$

$\Delta_c(K \to K')^{\mathcal{U}_p}(K)$ is equivalent with K' if $C(K) - K = \emptyset \Rightarrow C(K) \subseteq K$, but from the definition of closure it holds $K \subseteq C(K)$. So the proposition is correct if $C(K) \subseteq K \subseteq C(K) \Leftrightarrow K = C(K)$ i.e. when K is complete. $\diamond$

6.4 Semantic Identity, Reversibility and Composability Proofs

Prop. 8 If $K \sim K'$ then $\Delta_d(K \to K') = \Delta_c(K \to K') = \Delta_{dc}(K \to K') = \emptyset$.

Proof:

Recall the formula of Δ_d:

$\Delta_d(K \to K') = \{Add(t) \mid t \in K' - C(K)\} \cup \{Del(t) \mid t \in K - C(K')\}$

Assuming additions: $K' - C(K) \stackrel{(Def1)}{=} \emptyset$, assuming deletions: $K - C(K') \stackrel{(Def1)}{=} \emptyset$. This means that $\Delta_d(K \to K') = \emptyset$.

Recall the formula of Δ_c:

$\Delta_c(K \to K') = \{Add(t) \mid t \in C(K') - C(K)\} \cup \{Del(t) \mid t \in C(K) - C(K')\}$

Assuming additions: $C(K') - C(K) \stackrel{(Def\ 1)}{=} \emptyset$, assuming deletions: $C(K) - C(K') \stackrel{(Def1)}{=} \emptyset$. This means that $\Delta_c(K \to K') = \emptyset$.

Finally, recall the formula of Δ_{dc}:

$\Delta_{dc}(K \to K') = \{Add(t) \mid t \in K' - C(K)\} \cup \{Del(t) \mid t \in C(K) - C(K')\}$

Assuming additions: $K' - C(K) \stackrel{(Def1)}{=} \emptyset$, assuming deletions: $C(K) - C(K') \stackrel{(Def1)}{=} \emptyset$. This means that $\Delta_{dc}(K \to K') = \emptyset$. $\diamond$

Prop. 9 If $K \sim K'$, $\{K, K'\} \subseteq \Psi$ and $RF(K), RF(K')$ then $\Delta_e(K \to K') = \emptyset$

Proof:

$K \sim K' \Rightarrow C(K) = C(K')$. We know that $RF(K), RF(K')$ and $\{K, K'\} \subseteq \Psi$. From the above relations we can conclude that $K = K'$

Recall that $\Delta_e(K \to K') = \{Add(t) \mid t \in K' - K\} \cup \{Del(t) \mid t \in K - K'\}$

Assuming additions we have: $K' - K = \emptyset$, assuming deletions we have: $K - K' = \emptyset$. This means that $\Delta_e(K \to K') = \emptyset$. $\diamond$

Prop. 10 If $K \sim K'$, and (a) $K' \subseteq K$ or (b) $\{K, K'\} \subseteq \Psi$ and $RF(K), RF(K')$ then $\Delta_{ed}(K \to K') = \emptyset$

Proof:

Recall that $\Delta_{ed}(K \to K') = \{Add(t) \mid t \in K' - K\} \cup \{Del(t) \mid t \in K - C(K')\}$. We know that $K - C(K') \stackrel{(Def1)}{=} \emptyset$. Assuming the (a) premises of the proposition we have $K' \subseteq K \Rightarrow K' - K = \emptyset$. This means that $\Delta_{ed}(K \to K') = \emptyset$. Assuming the (b) premises of the proposition we have $K \sim K' \Rightarrow C(K) = C(K')$. We know that $RF(K), RF(K')$ and $\{K, K'\} \subseteq \Psi$. From the above relations we can conclude that $K = K'$, so $K' - K = \emptyset$. This also means that $\Delta_{ed}(K \to K') = \emptyset$. $\diamond$

Prop. 11 For every pair of valid knowledge bases K, K' it holds:

$$Inv(\Delta_e(K \to K')) = \Delta_e(K' \to K)$$
$$Inv(\Delta_c(K \to K')) = \Delta_d(K' \to K)$$
$$Inv(\Delta_d(K \to K')) = \Delta_c(K' \to K)$$

Proof:

The following hold for Δ_e:

$$\begin{aligned} Inv(\Delta_e(K \to K')) &= \{Add(t) \mid t \in K - K'\} \cup \{Del(t) \mid t \in K' - K\} \\ \Delta_e(K' \to K) &= \{Add(t) \mid t \in K - K'\} \cup \{Del(t) \mid t \in K' - K\} \end{aligned}$$

This means that $Inv(\Delta_e(K \to K')) = \Delta_e(K' \to K)$.

The following hold for Δ_c:

$$\begin{aligned} Inv(\Delta_c(K \to K')) &= \{Add(t) \mid t \in C(K) - C(K')\} \cup \{Del(t) \mid t \in C(K') - C(K)\} \\ \Delta_c(K' \to K) &= \{Add(t) \mid t \in C(K) - C(K')\} \cup \{Del(t) \mid t \in C(K') - C(K)\} \end{aligned}$$

This means that $Inv(\Delta_c(K \to K')) = \Delta_c(K' \to K)$.

Finally, for Δ_d hold:

$$\begin{aligned} Inv(\Delta_d(K \to K')) &= \{Add(t) \mid t \in K - C(K')\} \cup \{Del(t) \mid t \in K' - C(K)\} \\ \Delta_d(K' \to K) &= \{Add(t) \mid t \in K - C(K')\} \cup \{Del(t) \mid t \in K' - C(K)\} \end{aligned}$$

This means that $Inv(\Delta_d(K \to K')) = \Delta_d(K' \to K)$. $\diamond$

Prop. 13 For any valid knowledge bases $K_1, K_2, K_3, ..., K_n$ it holds:

$$\begin{aligned}\Delta_e(K_1 \to K_2) \circ \Delta_e(K_2 \to K_3) \circ ... \circ \Delta_e(K_{n-1} \to K_n) &= \Delta_e(K_1 \to K_n) \\ \Delta_c(K_1 \to K_2) \circ \Delta_c(K_2 \to K_3) \circ ... \circ \Delta_c(K_{n-1} \to K_n) &= \Delta_c(K_1 \to K_n)\end{aligned}$$

Proof:

In order to prove the proposition we have to show that $\Delta_e(K_1 \to K_2) \circ \Delta_e(K_2 \to K_3)$ and $\Delta_e(K_1 \to K_3)$ produce the same add and del statements i.e.

$$\begin{aligned}\Delta^+ = (K_2 - K_1) \cup (K_3 - K_2) - (K_1 - K_2) \cup (K_2 - K_3) &= K_3 - K_1 \\ \Delta^- = (K_1 - K_2) \cup (K_2 - K_3) - (K_2 - K_1) \cup (K_3 - K_2) &= K_1 - K_3\end{aligned}$$

We start by proving some properties that are used to prove the proposition:

$(K_2 - K_1) - (K_1 - K_2) \cup (K_2 - K_3) \overset{(g)}{=} ((K_2 - K_1) - (K_1 - K_2)) \cap ((K_2 - K_1) - (K_2 - K_3)) =$
$(K_2 - K_1) \cap ((K_2 - K_1) - (K_2 - K_3)) \overset{(c)}{=} (K_2 - K_1) \cap (((K_2 - K_1) - K_2) \cup ((K_2 - K_1) \cap K_3)) =$
$(K_2 - K_1) \cap (K_2 - K_1) \cap K_3 = (K_2 - K_1) \cap K_3 \overset{(e)}{=} (K_3 \cap K_2) - K_1 \quad$ (h)

Another property that holds is:

$(K_3 - K_2) - (K_1 - K_2) \cup (K_2 - K_3) \overset{(g)}{=} ((K_3 - K_2) - (K_1 - K_2)) \cap ((K_3 - K_2) - (K_2 - K_3)) =$
$((K_3 - K_2) - (K_1 - K_2)) \cap (K_3 - K_2) \overset{(c)}{=} (((K_3 - K_2) - K_1) \cup ((K_3 - K_2) \cap K_2)) \cap (K_3 - K_2) =$
$((K_3 - K_2) - K_1) \cap (K_3 - K_2) = (K_3 - K_2) - K_1 \quad$ (j)

Moreover it holds:

$(K_1 - K_2) - (K_2 - K_1) \cup (K_3 - K_2) \overset{(g)}{=} ((K_1 - K_2) - (K_2 - K_1)) \cap ((K_1 - K_2) - (K_3 - K_2)) =$
$(K_1 - K_2) \cap ((K_1 - K_2) - (K_3 - K_2)) \overset{(c)}{=} (K_1 - K_2) \cap (((K_1 - K_2) - K_3) \cup ((K_1 - K_2) \cap K_2)) =$
$(K_1 - K_2) \cap ((K_1 - K_2) - K_3) = (K_1 - K_2) - K_3 \quad$ (k)

Finally, it holds:

$(K_2 - K_3) - (K_2 - K_1) \cup (K_3 - K_2) \overset{(g)}{=} ((K_2 - K_3) - (K_2 - K_1)) \cap ((K_2 - K_3) - (K_3 - K_2)) =$
$((K_2 - K_3) - (K_2 - K_1)) \cap (K_2 - K_3) \overset{(c)}{=} (((K_2 - K_3) - K_2) \cup ((K_2 - K_3) \cap K_1)) \cap (K_2 - K_3) =$
$(K_2 - K_3) \cap K_1 \cap (K_2 - K_3) = (K_2 - K_3) \cap K_1 \overset{(e)}{=} (K_1 \cap K_2) - K_3 \quad$ (m)

Using the above equalities we can prove the proposition:

$\Delta^+ = (K_2 - K_1) \cup (K_3 - K_2) - (K_1 - K_2) \cup (K_2 - K_3) \overset{(f)}{=} ((K_2 - K_1) - (K_1 - K_2) \cup (K_2 -$

$K_3)) \cup ((K_3 - K_2) - (K_1 - K_2) \cup (K_2 - K_3)) \overset{(h),(j)}{=} ((K_3 \cap K_2) - K_1) \cup ((K_3 - K_2) - K_1) \overset{(f)}{=}$
$((K_3 \cap K_2) \cup (K_3 - K_2)) - K_1 = K_3 - K_1$

$\Delta^- = (K_1 - K_2) \cup (K_2 - K_3) - (K_2 - K_1) \cup (K_3 - K_2) \overset{(f)}{=} ((K_1 - K_2) - (K_2 - K_1) \cup (K_3 -$
$K_2)) \cup ((K_2 - K_3) - (K_2 - K_1) \cup (K_3 - K_2)) \overset{(k),(m)}{=} ((K_1 - K_2) - K_3) \cup ((K_1 \cap K_2) - K_3) \overset{(f)}{=}$
$((K_1 - K_2) \cup (K_1 \cap K_2)) - K_3 = K_1 - K_3$

We have proven that Δ_e satisfies composability for two deltas. Successively, it is easy to prove that it satisfies composability for n deltas.

$$\begin{aligned}
&\Delta_e(K_1 \to K_2) \circ \Delta_e(K_2 \to K_3) \circ ... \circ \Delta_e(K_{n-1} \to K_n) &&= \\
&\Delta_e(K_1 \to K_3) \circ \Delta_e(K_3 \to K_4) \circ ... \circ \Delta_e(K_{n-1} \to K_n) &&= \\
&\Delta_e(K_1 \to K_4) \circ \Delta_e(K_4 \to K_5) \circ ... \circ \Delta_e(K_{n-1} \to K_n) &&= \quad ... = \Delta_e(K_1 \to K_n)
\end{aligned}$$

The proof for Δ_c is similar to Δ_e with the difference that $C(K_1), C(K_2)$ and $C(K_3)$ are used instead of K_1, K_2 and K_3. So the proof for Δ_c is omitted. $\diamond$

6.5 Proofs for streaming execution of deltas

Prop. 14 For every $M \subseteq \mathcal{S}$ produced by Δ_d, Δ_{dc} and Δ_c, Alg. 1 terminates

Proof

Let Y be the satisfiable deletions and Z the unsatisfiable deletions at any point during the execution of the algorithm (i.e. the sets Sat and $UnSat$ respectively). We have to prove that whenever $|Y| = 0$ we also have $|Z| = 0$ this means that our algorithm always terminates since all elements of M are satisfied.

Both Δ_c and Δ_{dc} produce the following set of delete statements: $X = \{\ Del(t') \mid t' \in C(K) - C(K')\ \}$. An element $Del(t)$ will be satisfied if $t \in R(K)$. So the set Y, i.e. the satisfiable deletions of X, is defined as $Y = R(K) \cap (C(K) - C(K')) = R(K) - C(K')$. Let's now define Z. Recall that a $Del(t')$, may not be satisfied (when applied to K) only if $t' \in C(K) - R(K)$. So the set Z, i.e. the unsatisfiable deletions of X, is defined as $Z = (C(K) - R(K)) \cap (C(K) - C(K')) =$

$C(K) - (R(K) \cup C(K'))$.

Let's now investigate whether $|Y| = 0 \Rightarrow |Z| = 0$ holds. At first, notice that $Y = \emptyset \Leftrightarrow R(K) - C(K') = \emptyset \Leftrightarrow R(K) \subseteq C(K')$. Also note that $R(K) \subseteq C(K') \Rightarrow C(K) \subseteq C(K')$. This is based on the properties of the closure operator: if we have two sets A and B such that $A \subseteq B$ and B is closed with respect to the closure operator C (i.e. $C(B) = B$), then $C(A) \subseteq B$.

Returning to our problem, if $Y = \emptyset$ (that is if $R(K) \subseteq C(K')$), then the formula $Z = C(K) - (R(K) \cup C(K'))$ is equivalent to $Z = C(K) - C(K')$. But above we have seen that $Y = \emptyset \Rightarrow C(K) \subseteq C(K')$ too. It follows that $Z = \emptyset$. So the algorithm always terminates.

The above is actually the proof of the proposition: If $|R(K) - C(K')| = 0$ then $|C(K) - (R(K) \cup C(K'))| = 0$.

The proof for Δ_d is similar. $\diamond$

Prop. 19 If $t \in K$ and t is not a redundant triple i.e. $t \in R(K)$ then
$C(C(K) - \{t\}) = C(K) - \{t\}$.

Proof:

As t is not a redundant triple, t cannot be inferred. i.e. if $t = (a, b) \Rightarrow \nexists c : \{(a, c), (c, b)\} \subseteq K$. The operator C has an effect (adds something) only if there are elements of the form $\{(a, c), (c, b)\}$ and the element (a, b) is missing, then it returns $\{(a, c), (c, b), (a, b)\}$. However $C(K) - \{t\}$ does not contain any elements of the form $\{(a, c), (c, b)\}$ with the element (a, b) missing because t (i.e. (a, b)) is not inferred. So the second C (i.e. $C(C(K) - \{t\})$) is void. This means that $C(C(K) - \{t\}) = C(K) - \{t\}$.$\diamond$

Prop. 20 If $t_1 \in R(K)$ then $R(C(R(C(K) - \{t_1\})) - \{t_2\}) = R(C(K) - \{t_1, t_2\})$

Proof:
$R(C(R(C(K) - \{t_1\})) - \{t_2\}) \stackrel{(b)}{=} R(C(C(K) - \{t_1\}) - \{t_2\}) \stackrel{(prop.19)}{=} R(C(K) - \{t_1\} - \{t_2\}) = R(C(K) - \{t_1, t_2\})$

Let $\{Del(t_1), Del(t_2)\} \subseteq \Delta_x(K \rightarrow K')$. This proposition tells us that it equivalent to compute the reduction once in the end, and to execute each deletion exactly as stated in $\mathcal{U}_{ir}$-semantics, i.e. one reduction after each operation. This is true under the assumption that the triples that exist at the reduction are deleted first. Recall that our execution algorithm (Alg. 1) for sequence of change operations will never delete a redundant triple (it first deletes the triples that exist in the reduction).

In our case it has been proved that there is an order of $Del(t)$ operations that satisfy these requirements.

This proposition is generalized for more than 2 triples deleted:
$R(C(...R(C(R(C(K) - \{t_1\})) - \{t_2\})...) - \{t_n\}) = R(C(K) - \{t_1, t_2, ..., t_n\})$ $\diamond$

Prop. 21 If $K \subseteq \Psi$, then $R(R(K) \cup \{t_1\}) = R(K \cup \{t_1\})$

Proof:

Let $IK = K - R(K)$ i.e. all the redundant triples of K. A triple $t \in IK$ will remain redundant after adding a triple t_1 to K i.e. the addition of a triple cannot make a redundant triple non-redundant. Let IK' be the triples that became redundant after adding t_1 to K. At the formula $R(R(K) \cup \{t_1\})$ the redundant triples that exist in IK (i.e. $t \in IK$) are removed first and next, then the redundant triples that exist in IK' (i.e. $t \in IK'$) are removed. In formula $R(K \cup \{t_1\})$ all the redundant triples (i.e. $t \in IK \cup IK'$) are removed at one step. In both cases the result is the same.$\diamond$

Prop. 22 If $K \subseteq \Psi$, then $R(R(K \cup \{t_1\}) \cup \{t_2\}) = R(K \cup \{t_1, t_2\})$

Proof:

Let $K' = K \cup \{t_1\}$. We can rewrite the proposition as:
$R(R(K \cup \{t_1\}) \cup \{t_2\}) = R(K \cup \{t_1, t_2\}) \Leftrightarrow R(R(K') \cup \{t_2\}) = R((K \cup \{t_1\}) \cup \{t_2\}) \Leftrightarrow$
$R(R(K') \cup \{t_2\}) = R(K' \cup \{t_2\}) \overset{(prop.21)}{\Leftrightarrow} R(K' \cup \{t_2\}) = R(K' \cup \{t_2\})$

Generalizing the proposition for n triples we can write:
$R(...R(R(K \cup \{t_1\}) \cup \{t_2\}) ... \cup \{t_n\}) = R(K \cup \{t_1, t_2, ..., t_n\})\diamond$

Prop. 23 Let $A = \{t_{1a}, t_{2a}, ..., t_{ma}, \}$, $D = \{t_{1d}, t_{2d}, ..., t_{nd}\}$. If $K \subseteq \Psi$ and $t_{id} \in R(K_i)$ where K_i is the result of the execution of the i deletion, then
$R(...R(R(R(C(...R(C(R(C(K) - \{t_{1d}\})) - \{t_{2d}\})...) - \{t_{nd}\}) \cup t_{1a}) \cup t_{2a}) ... \cup t_{na}) = R((C(K) - D) \cup A)$

Proof:

$R(...R(R(R(C(...R(C(R(C(K) - \{t_{1d}\})) - \{t_{2d}\})...) - \{t_{nd}\}) \cup t_{1a}) \cup t_{2a}) ... \cup t_{ma}) \overset{(prop20)}{=} R(...R(R(C(K) - D) \cup t_{1a}) \cup t_{2a}) ... \cup t_{ma}) \overset{(prop22)}{=} R((C(K) - D) \cup A)$

This means that the result is the same no matter whether the reduction and the closure are computed at every step or once in the end.$\diamond$

Theorem 4 If $\{K, K'\} \subseteq \Psi$ then for every delta produced by Δ_d, Δ_{dc} and Δ_c there exists at least one sequence of change operations that when executed under $\mathcal{U}_{ir}$ the result is equivalent to the execution of the corresponding *set* of change operations under $\mathcal{U}_{ir}$. Alg. 1 produces such a sequence.

Proof:

The Proposition 23 is the proof for the Theorem. $\diamond$