University of Crete
Computer Science Department

Managing the Specificity of Ontological Descriptions under Ontology Evolution

Mary Kampouraki
Master’s Thesis

Heraklion, February, 2011
ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΡΗΤΗΣ
ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΚΑΙ ΤΕΧΝΟΛΟΓΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΕΠΙΣΤΗΜΗΣ ΥΠΟΛΟΓΙΣΤΩΝ

Εξέλιξη Οντολογιών και Διαχείριση της Ειδικότητάς των Οντολογικών Περιγραφών

Εργασία που υποβλήθηκε από την
Μαίρη Καμπουράκη

ως μερική εκπόλυμωση των απαιτήσεων για την απόκτηση
METAPIΓΤΥΧΙΑΚΟΥ ΔΙΠΛΩΜΑΤΟΣ ΕΙΔΙΚΕΥΣΗΣ

Συγγραφέας:

__________________________________________________________
Μαίρη Καμπουράκη, Τμήμα Επιστήμης Υπολογιστών

Εισηγητική Επιτροπή:

__________________________________________________________
Γιάννης Τζίτζικας, Επίκουρος Καθηγητής, Επόπτης

__________________________________________________________
Αναστασία Αναλυτή, Ερευνήτρια του Ινστιτούτου Πληροφορικής ΙΤΕ, Μέλος

__________________________________________________________
Δημήτρης Πλεξουσάκης, Καθηγητής, Μέλος

Δεκτή:

__________________________________________________________
Άγγελος Μπίλας, Αναπληρωτής Καθηγητής
Πρόεδρος Επιτροπής Μεταπτυχιακών Σπουδών

Ηράκλειο, Φεβρουάριος 2011
Managing the Specificity of Ontological Descriptions under Ontology Evolution
Mary Kampouraki
Master’s Thesis
Computer Science Department, University of Crete

Abstract

Semantic Web Ontologies are not static but evolve as the understanding of the domain (or the domain itself) grows or evolves. This evolution happens independently of the ontological instance descriptions (for short metadata) which are stored in the various Metadata Repositories (MRs) or Knowledge Bases (KBs). However, it is a common practice for a MR/KB to periodically update its ontologies to their latest versions. This is done by migrating the available metadata to the latest version of the ontology. Usually such migrations are not difficult because new ontology versions are usually compatible with the past versions. However such migrations incur gaps regarding the specificity of migrated metadata. This results in inability to distinguish those metadata that should be reexamined for possible specialization (as consequence of the migration) from those for which this is not necessary. For this reason there is a need for principles, techniques, and tools that can manage the uncertainty incurred by such migrations, specifically techniques which can identify automatically the descriptions that are candidate for specialization, compute, rank and recommend possible specializations, and flexible interactive techniques for updating the metadata repository (and its candidate specializations), after the user (curator) accepts/rejects such recommendations. This problem is especially important for curated KBs which have increased quality requirements (e-Science).

This is the first work that elaborates on this problem. It formulates the problem, introduces the notion of extended KB consisting of the certain plus the possible (due to migration) specialized knowledge, and proposes principles and rules for updating it, assuming the RDF/S framework. Subsequently, it provides algorithms and reports experimental results (over real and synthetic datasets) demonstrating the feasibility of the approach. In addition, a compact representation of the possibilities is proposed for reducing the storage space requirements. Finally, it presents RIMQA (RDF Instance Migration Quality
Assistant), a tool which has been designed and implemented for supporting the entire lifecycle. To conclude, the proposed approach can enrich the lifecycle of curated Semantic Web data with quality management processes appropriate for scenarios where ontologies evolve frequently and independently from instance descriptions. As a consequence, this allows adopting iterative and agile ontology modeling approaches, appropriate for open environments like Linked Open Data (LOD).

**Supervisor:** Yannis Tzitzikas  
Assistant Professor
Εξέλιξη Οντολογιών και Διαχείριση της Ειδικότητας των Οντολογικών Περιγραφών

Μαίρη Καμπούρακη
Μεταπτυχιακή Εργασία
Τμήμα Επιστήμης Υπολογιστών, Πανεπιστήμιο Κρήτης

Περίληψη

Οι οντολογίες του Σημασιολογικού Ιστού δεν είναι στατικές αλλά εξελίσσονται για διάφορους λόγους, π.χ. λόγω εμπλουτισμού της εννοιοποίησης (conceptualization) του πεδίου εφαρμογής, ή της εξέλιξης του πεδίου εφαρμογής αυτού καθ’ εαυτόν. Αυτή η εξέλιξη γίνεται συνήθως ανεξάρτητα από τις οντολογικές περιγραφές (metadεδομένα) που είναι αποθηκευμένες στα διάφορα Αποθετήρια Μεταδεδομένων (Metadata Repositories) ή Βάσεις Γνώσεων (Knowledge Bases). Αποτελεί όμως κοινή πρακτική των MR/KB η περιοδική επικαιροποίηση των οντολογικών τους και αυτή η ανάγκη συνήθως αντιμετωπίζεται με τη μετανάστευση των ήδη εφαρμοσμένων οντολογικών περιγραφών στις νέες εκδόσεις των οντολογικών. Αυτή η μετάβαση συνήθως δεν έχει δυσκολίες αφού οι νέοτερες εκδόσεις στην πλειοψηφία τους είναι συμβατές με τις προηγούμενες. Παρά ταύτα, τέτοιες μεταναστεύσεις (ή γενικότερα ερμηνείες μεταδεδομένων βάση νεότερων εκδόσεων), δημιουργούν κενά σχετικά με την ειδικότητα (specificity) των περιγραφών. Αυτό οδηγεί σε αδυναμία διάκρισης των περιγραφών που επιδέχονται αναθεώρηση και πιθανής ειδύλλευσης, από εκείνες για τις οποίες δεν υπάρχει τέτοια ανάγκη. Για το λόγο αυτό απαιτούνται αρχές, μηχανισμοί και εργαλεία που να μπορούν αυτόματα να διαχειριστούν την αβεβαιότητα που προκύπτει από τέτοιες μεταναστεύσεις, συγκεκριμένα τεχνικές που να εντοπίζουν αυτόματα τις περιγραφές που επιδέχονται εξειδίκευση (κατόπιν μετανάστευσης), να υπολογίζουν/ κατατάσσουν και συστήνουν τις πιθανές εξειδικεύσεις τους, καθώς επίσης και ευέλικτες διαλογικές τεχνικές ενημέρωσης της βάσης περιγραφών (και των πιθανών εξειδικεύσεων τους) κατόπιν αποδοχής/απόρριψης των συστάσεων από τον επιμελητή του αποθετηρίου. Το πρόβλημα αυτό είναι σημαντικό για τις επιμελημένες (curated) Βάσεις Γνώσεων, οι οποίες έχουν αυξημένες απαιτήσεις ποιότητας (π.χ. βάσεις επιστημονικών δεδομένων).

Η παρούσα εργασία είναι η πρώτη που ασχολείται με αυτό το πρόβλημα. Αρχικά διατυπώνει τυπικά το πρόβλημα, εισάγει την έννοια της διευρυμένης βάσης γνώσεων (extended KB)
αποτελούμενη από τη σήγουρη γνώση και τις πιθανές (λόγω των μεταναστεύσεων) εξειδικεύσεις της, και προτείνει αρχές και κανόνες που πρέπει να διέπουν την ενημέρωσή της, επεκτεντωνόμενα στο πλαίσιο RDF/S. Εν συνεχεία, διδονται οι σχετικοί αλγόριθμοι, αποδεικνύοντας τυπικά η ορθότητά τους και αναφέρονται πειραματικά αποτελέσματα επί πραγματικών και συνθετικών δεδομένων. Συνάμα, και με στόχο τη μείωση του απαιτούμενου αποδημικού χώρου, προτείνεται μια συμπαγής αναπαράσταση των πιθανών ειδικεύσεων. Τέλος, παρουσιάζεται ένα εργαλείο που σχεδιάστηκε και αναπτύχθηκε για τη στήριξη όλης της διαδικασίας, ονόματι RIMQA (RDF Instance Migration Quality Assistant).

Εν καταλείδι, η προτεινόμενη προσέγγιση μπορεί να εμπλουτίσει τον κύκλο ζωής των επιμελημένων (curated) (meta)δεδομένων του Σημασιολογικού Ιστού με διαδικασίες διαχείρισης ποιότητας, κατάλληλες για σενάρια όπου οι οντολογίες εξελίσσονται συχνά και συνεξαρτητικά από τις περιγραφές βάσει αυτών. Η προσέγγιση αυτή συνάμα επιτρέπει την υιοθέτηση επαναληπτικών προσεγγίσεων μοντελοποίησης οντολογιών, οι οποίες είναι κατάλληλες για ανοιχτά περιβάλλοντα όπως τα Διασυνεδρεμένα Ανοιχτά Δεδομένα (Linked Open Data).

Επόπτης Καθηγητής: Γιώργος Τζίτζικας
Επίκουρος Καθηγητής

8
Ολοκληρώνοντας τη μεταπτυχιακή μου εργασία, θα ήθελα να ευχαριστήσω όλους όσους με στήριζαν και ήταν κοντά μου στη διάρκεια των μεταπτυχιακών σπουδών.

Αρχικά, θα ήθελα να ευχαριστήσω θερμά τον επίκουρο καθηγητή του τμήματος Επιστήμης Υπολογιστών του Πανεπιστημίου Κρήτης, ερευνητή στο Εργαστήριο Πληροφορικής Ψυχογιατρείων (ΕΠΣ) του Ινστιτούτου Πληροφορικής του Ιδρύματος Πληροφορικής του Φοιτητικού Πανεπιστημίου Κρήτης και υπότροφο της μεταπτυχιακής μου εργασίας, Δρ. Γιάννη Τζίτζικα, για την εξαιρετική συνεργασία που είχαμε τα τελευταία δύο χρόνια και για την ουσιαστική συμβολή του στην εξέλιξη αυτής της εργασίας. Η καθοδήγησή του και η εμπιστοσύνη που έδειχνε στο πρόσωπό μου με βοήθησαν σημαντικά σε όλη τη διάρκεια της μεταπτυχιακής μου εργασίας. Επίσης, θα ήθελα να ευχαριστήσω από οργάνη την ερευνητή της ΕΠΣ του ΙΤΕ-ΙΠ, Δρ. Αναστασία Αναλυτή, για την άριστη συνεργασία που είχαμε τον τελευταίο χρόνο, για τη συμβολή της στην θεωρητική θεμελίωση αυτής της εργασίας, για την επιμέλεια των αποδείξεων και για την πολύτιμη βοήθεια και στήριξη που μου παρείχε. Ακόμη, θα ήθελα να ευχαριστήσω τον καθηγητή του τμήματος Επιστήμης Υπολογιστών του Πανεπιστημίου Κρήτης και υπότροφο της μεταπτυχιακής εργασίας, Δρ. Δημήτρη Πλεξουσάκη, για την προθυμία του να είναι στην τριμελή εξεταστική επιτροπή αυτής της μεταπτυχιακής εργασίας. Επίσης, θα ήθελα να ευχαριστήσω τον Επιτρόπο της ΕΠΣ του ΙΤΕ-ΙΠ για τη χορήγηση υποτροφίας στα χρόνια της μεταπτυχιακής μου εργασίας.

Χωρίς την υποστήριξη της οικογένειάς μου δε θα είχα τη δυνατότητα να αντεπεξέλθω σε όλες τις δυσκολίες που παρουσιάστηκαν κατά τη διάρκεια των σπουδών μου. Οφείλω ένα μεγάλο ευχαριστώ στη μητέρα μου, Φωτεινή, που είναι πάντα δίπλα μου και με προσφέρει όλες τις επιλογές μου και με παρακολουθεί και στήριξης και ενθάρρυνται στη μεταπτυχιακή μου εργασία.

Θα ήθελα ακόμη να ευχαριστήσω όλους αυτούς τους ψύλλους και συμφοιτητές μου, που ήταν κοντά μου στα φοιτητικά χρόνια, για τις όμορφες στιγμές που έγιναν μαζί και για τις ανεκτίμητες αναμνήσεις που μου χάρισαν. Ειδικότερα, θα ήθελα να ευχαριστήσω τη συμμετοχή του μου και την εμπιστοσύνη της της μεταπτυχιακής μου εργασίας. Επίσης, θα ήθελα να ευχαριστήσω όλους τους ΕΠΣ του ΙΤΕ-ΙΠ για τη γνωριμία με τους και για το υπεράνοια σταθερό περιβάλλον εργασίας τα τελευταία δύο χρόνια.

Τέλος, θα ήθελα να ευχαριστήσω τον Βαγγέλη Γογολίδη για την υπομονή του και την απεριόριστη στήριξη και ενθάρρυνση που συνεργαζόμασταν στην επιμέλεια της μεταπτυχιακής μου εργασίας.
# Contents

Table of Contents ii

List of Tables iii

List of Figures vi

1 Introduction 1

1.1 Investigating the Problem ........................................... 3
  1.1.1 Motivating Example ............................................. 4
  1.1.2 Supporting the Full Life Cycle ................................. 4
  1.1.3 The Difficulties ................................................. 5
  1.1.4 On MSA, RDF, and Open/Closed World Assumptions .......... 6

1.2 Real World Ontologies ............................................... 6

1.3 Contribution of this thesis ......................................... 10

1.4 Organization of this thesis ......................................... 11

2 Background 13

3 The Notion of X-partition 17

4 Transition of X-partitions 23

5 On Managing X-partitions without $M$-sets 31
  5.1 Algorithmic Perspective ........................................... 33

6 Specificity Life Cycle Management 41
  6.1 Ranking Possibilities ............................................. 45
List of Tables

2.1 Schema and Instance Triples ................................................. 15

9.1 $|C_i|$, $|Pr_i|$, $|I_i|$ and $|S_i|$ for each $i$ Music Ontology version .......... 85
9.2 $|P_{K'}|$ Sizes for Music Ontology migrations. $|P_K| = 0$ .................. 86
9.3 $|P_{K'}|$ Sizes for Music Ontology migrations. $|P_K| \neq 0$ .................. 86
9.4 Execution Times (in msec) for Music Ontology migrations for $|P_K| = 0$ . . 87
9.5 Execution Times (in msec) for Music Ontology migrations for $|P_K| \neq 0$ . . 87
9.6 $|C_i|$, $|Pr_i|$, $|I_i|$ and $|S_i|$ for each $i$ version of the Synthetic data set . . . . . 89
9.7 $|P_{K'}|$ Sizes for Synthetic Data migrations. $|P_K| = 0$ .................. 89
9.8 $|P_{K'}|$ Sizes for Synthetic Data migrations. $|P_K| \neq 0$ .................. 89
9.9 Execution Times (in sec) for Synthetic Data migrations for $|P_K| = 0$ .... 90
9.10 Execution Times (in sec) for Synthetic Data migrations for $|P_K| \neq 0$ .... 90

1 Symbols and Description ...................................................... 130
List of Figures

1.1 Motivating example .................................................. 4
1.2 The process of exploiting possibilities ............................... 5
1.3 An example of LCSH weekly update ................................. 8
1.4 A subsequent weekly update of that shown in Figure 1.3 ............ 9

4.1 An instance migration scenario ...................................... 24
4.2 X-partition transitions after successive migrations ............... 25
4.3 Motivating example for Postulate II2 ............................... 26

5.1 Second Instance Migration Scenario ................................. 32
5.2 Third Instance Migration Scenario ................................. 37
5.3 Fourth Instance Migration Scenario ................................. 37
5.4 Fifth Instance Migration Scenario .................................. 38

6.1 Current and Proposed Migration Process ............................ 42
6.2 Example of the Possibility Resolution Process ..................... 44
6.3 Ranking possible class instance triples ............................. 48

7.1 First scenario for Extended Possibilities ......................... 54
7.2 Second scenario for Extended Possibilities ....................... 55
7.3 Ranking Composite Possibilities .................................. 56

8.1 Certain Modification Inheritance Rules ........................... 65
8.2 Non-Backwards Compatible Schema Evolution ................... 68

9.1 RIMQA: Select to create a new RIMQA project ..................... 72
9.2 RIMQA: Create a new RIMQA project and start the Curation Process . 72
9.3 RIMQA: Statistics ......................................................... 74
9.4 RIMQA: Curate the Instance Descriptions .......................... 74
9.5 RIMQA: Show all Possible Class Instance Triples ................. 75
9.6 RIMQA: Select a Possible Instance (by its URI) ................. 76
9.7 RIMQA: Select a Possible Instance (by its Possible Classes) .... 77
9.8 RIMQA: View Possible Classes of the selected Instance ......... 78
9.9 RIMQA: View Composite Possibilities associated with the selected Instance 79
9.10 Object-centered Compact Storage Policy for Possible Class Instance Triples 81
9.11 Object-centered Compact Storage Policy for Possible Property Instance Triples .................................................. 81

10.1 Different sequences of transitions ...................................... 94
10.2 Different sequences of non-backwards compatible transitions ...... 96
Chapter 1

Introduction

Semantic Web is a group of methods and technologies to allow machines to understand the meaning - or “semantics” - of information on the World Wide Web. The term was coined by World Wide Web Consortium (W3C) director Tim Berners-Lee [20]. He defines the Semantic Web as “a web of data that can be processed directly and indirectly by machines.” While the term “Semantic Web” is not formally defined, it is mainly used to describe the model and technologies proposed by the W3C. These technologies include the Resource Description Framework (RDF), a variety of data interchange formats (e.g. RDF/XML, N3, Turtle, N-Triples), and notations such as RDF Schema (RDFS) and the Web Ontology Language (OWL), all of which are intended to provide a formal description of concepts, terms, and relationships within a given knowledge domain. As already mentioned, the Resource Description Framework (RDF) is a family of World Wide Web Consortium (W3C) specifications originally designed as a metadata data model. It has come to be used as a general method for conceptual description or modeling of information that is implemented in web resources, using a variety of syntax formats.

In computer science and information science, an ontology is a formal representation of knowledge as a set of concepts within a domain, and the relationships between those concepts. It is used to reason about the entities within that domain, and may be used to describe the domain. In theory, an ontology is a “formal, explicit specification of a shared conceptualization” [10]. An ontology provides a shared vocabulary, which can be used to model a domain, that is, the type of objects and/or concepts that exist, and their
properties and relations. Ontologies have been used in several domains such as, Artificial Intelligence, Semantic Web, Configuration Systems, Systems Engineering, Software Engineering, Information Retrieval, Conceptual Modeling, Library Science, Enterprise Modeling, e-Learning, e-Government, e-Commerce, Biomedical Informatics, Natural Language Processing, and Information Architecture as a form of knowledge representation about the world or some part of it. The creation of domain ontologies is also fundamental to the definition and use of an enterprise architecture framework.

As we mentioned above, ontologies provide a shared conceptualization of a domain by defining the concepts in the domain and describing how those concepts are related to each other. Several reasons for changing an ontology have been identified in the literature [8]: an ontology may need to change because it offers a richer conceptualization of the problem domain, the domain of interest has changed, the perspective under which the domain is viewed has changed, or the user/application needs have changed.

An important observation is that this evolution happens independently of the ontological instance descriptions which are stored in the various Metadata Repositories (MRs) or Knowledge Bases (KBs). With the term ontological instance description, we refer to RDF/S [2] descriptions that classify an instance $o$ to a class $c$ or relate two instances $o, o'$ with a property $p_r$. With the term MR or KB, we refer to a stored corpus of ontological instance descriptions. They can be stored in files or in RDF/S databases (i.e. RDF triple-stores [23]). The evolution of ontologies\(^1\) happens independently of the ontological instance descriptions due to the distributed nature of the Web and the Semantic Web. For instance, this is the case with ontologies maintained by standardization authorities. However, it is a common practice (mainly for interoperability purposes) for a KB to periodically update its ontologies to their latest versions by “migrating” the stored instance descriptions to the latest ontology versions. Such migrations are usually not difficult, because newer versions are mainly (or constructed to be) compatible with past ones. Nevertheless, they incur gaps regarding the specificity of the migrated instance descriptions, i.e. inability to distinguish those that should be reexamined (for possible specialization as consequence of the migration) from those for which no reexamination is justified. It follows that quality control is very laborious and error-prone.

\(^1\)In this work, by the term of ontology we refer only to schema information.
1.1 Investigating the Problem

To start with, consider a corpus of instance descriptions and suppose that at certain points in time we can make the assumption that the available instance descriptions are the most specific and detailed descriptions that are possible with respect to the employed ontology. For instance, we can make such an assumption after explicit human (e.g. by the curator of the KB) inspection and verification [3], or in cases where the descriptions have been produced automatically by a method that is guaranteed to produce specific descriptions (e.g. by transforming curated relational data to RDF/S descriptions [28], or by automatic classification to categories each defined by sufficient and necessary conditions, etc.). We will hereafter refer to this assumption by the name maximum specificity assumption (for short MSA). It is not hard to see that if the new version of the ontology is richer than the past one, then the corpus of the migrated instance descriptions may no longer satisfy the MSA with respect to the new ontology. This thesis elaborates on this problem.

The ability to identify the instance descriptions that satisfy the MSA and those that do not, is useful in order to address questions of the form: (a) for what descriptions can we make the MSA? (b) what (class or property) instances should probably be reclassified (to more refined classes or properties), and (c) which are the candidate new classes or properties (refinements) of such instances? The above questions are very useful for curating a corpus of instance descriptions, i.e. for managing its specificity as the corpus evolves over time. Without special support, such tasks would be unacceptably expensive and vulnerable to omissions, for large datasets. Just indicatively, CIDOC CRM\(^2\) is one ontology (expressed in RDF/S) which is used by several ongoing EU projects, and it is curated (i.e. extended) by an authority (community) that is different from the various communities, curators of repositories, or simple users who keep creating instance descriptions with respect to that ontology. In practice, whenever a new version appears, the available instance descriptions are migrated to that version, and it is worth noting that this ontology has been revised at least 5 times the last two years (one recent version is described at [33]).

\(^2\)CIDOC CRM (ISO 21127) is a core ontology describing the underlying semantics of data schemata and structures from all museum disciplines and archives (its RDF representation contains 78 classes and 250 properties from which 7 are literal-valued) (available from http://www.cidoc-crm.org/).
1.1.1 Motivating Example

We will explain the main idea of our approach using the toy example depicted at Figure 1.1. Consider an e-commerce portal that sells various kinds of products. Suppose a car $c_1$ that has been classified under the class Car, and a person $p_1$ that has been classified under the class Person, defined in an ontology Ont1, and suppose that both classes have no subclasses. Assume that for the current set of instance descriptions according to Ont1 the MSA holds (i.e. they are complete with respect to specificity). Thus, $c_1$ is not a Person and $p_1$ is not a Car. Let Ont2 be a new version of that ontology, which among others, defines the subclasses of the classes Car and Person, shown at Figure 1.1 (right).

![Motivating example](image)

Figure 1.1: Motivating example

All subclasses of Car are possible classes for $c_1$. Adult is not a possible class for $c_1$, since $c_1$ was not a person according to Ont1. None of the subclasses of Car is a possible class for $p_1$, since $p_1$ was not a car according to Ont1. Moreover, notice that Ont1 defines a property owns and suppose that ($p_1$ owns $c_1$) is an instance description. Also notice that Ont2 defines a subproperty sells of owns between Person and Car. This property will be prompted as a possible specialization of the association between $p_1$ and $c_1$.

1.1.2 Supporting the Full Life Cycle

Furthermore, apart from identifying the information that could be further specialized, we would like to aid making it as specific as possible. Therefore, we should support flexible and interactive processes for managing the computed possibilities, where the user will be able to either accept or reject the computed recommendations, and eventually update the knowledge base reaching to a state where the MSA holds. The ranking of possibilities is important for designing user-friendly interaction schemes. We propose a process like the
one sketched in Figure 1.2 (described in detail in Chapter 6). Specifically, assume that the user selects some instances then the system displays ranked all or some of the possible instance descriptions for the selected instances. The user accepts or rejects these instance descriptions and the system updates appropriately the KB and its possible part. Note that the possible part of the KB is stored explicitly and separately. Returning to the example shown in Subsection 1.1.1, this means that we can rank the possible classes for \( c_1 \), so that if the user is prompted to select a possible class for \( c_1 \), then Diesel and Ecological will be the first classes to be displayed. If the user rejects the class Ecological, then all its subclasses will be rejected from the possible classes (and this reduces the effort required for reaching a state where the MSA holds).

1.1.3 The Difficulties

In real cases the computation of possibilities is more complex than the case of the example shown in Subsection 1.1.1, as we can have conflicts among (a) new positive knowledge inferable from the instance descriptions and the new schema, (b) new “negative” information inferable from the past negative instance descriptions and the new schema, and (c) the previously computed possible instance descriptions (possible refinements). In fact, our approach resolves such conflicts by considering that (a) has higher priority than (b), and (b) has higher priority than (c).

In addition, it should be possible to update correctly the set of possibilities, at scenarios with several successive instance migrations interwoven with several (positive or negative) user feedbacks. Finally, another challenge is to reduce the information that has to be kept
to support this scenario. Specifically to avoid having to keep negative information of any kind, and to devise compact representations for the possibilities.

1.1.4 On MSA, RDF, and Open/Closed World Assumptions

We should clarify that we do not violate the Open World Assumption of RDF/S. It is the MSA that allows us to infer the negative knowledge of the previous example. Only if we explicitly make the MSA we can then exploit it (in a Closed World Assumption manner) in the context of ontology evolution for formalizing the way possibilities are defined. To clarify that we can also capture the Open World Assumption of RDF, suppose that we start from an RDF/S KB for which we know nothing regarding its completeness or specificity. We can capture this case by considering that every instance description, that can be formed using the ontology and is not certain, is possible. That is, the set of “negative assertions” is empty (in our example that would mean that Adult can be a possible class for c1 and Car can be a possible class for p1). Then, we can still use and exploit our machinery when we migrate our descriptions to subsequent schema versions, and the steps of the life cycle that we propose can produce the “negative” statements. If however one knows that one particular RDF/S KB is complete (regarding specificity), which can be true in the context of curated knowledge bases, then he can “apply” the MSA. This means that every instance description, that can be formed using the ontology and is not certain, is negative. Thus, the set of possibilities is empty (in our example that would mean that Adult is not a possible class for c1 and Car is not a possible class for p1).

1.2 Real World Ontologies

The proposed instance description quality management can be useful in several scientific domains. Below we present the most prevalent ones along with ontologies which capture the knowledge of these fields.

- Digital Libraries: Library of Congress Subject Headings (LCSH) comprise a thesaurus (in the information technology sense) of subject headings, maintained by
the United States Library of Congress, for use in bibliographic records. LC Subject Headings are an integral part of bibliographic control, which is the function by which libraries collect, organize and disseminate documents. The Subject Headings are published in large red volumes (currently five), which are typically displayed in the reference sections of research libraries. They may also be searched online in the Library of Congress Classification Web⁢³, a subscription service, or free of charge (as individual records) at Library of Congress Authorities. The Library of Congress issues weekly updates⁣⁴ (see in Figures 1.3 and 1.4 two parts of subsequent weekly updates). The data is published for a fee by the Cataloging Distribution Service. A change log can be found in “https://addons.omeka.org/trac/log/plugins/Lcsh”. LCSH Authority Records have MARC/XML or MADS/XML format. However, they can be translated into RDF documents according to the SKOS⁤⁵ project’s Quick Guide to Publishing a Thesaurus on the Semantic Web.

• e-Government: oeGOV⁶ is making and publishing W3C OWL ontologies for e-Government. The oeGOV ontologies are OWL models of the Organizational structure of government, the FEA models and QUDT (Quantities, Units, Dimensions and Data Types). Ontologies expressed in OWL allow data to be interpreted and aggregated across the web. By having foundation ontologies of eGovernment we enable a web of government data. The first step is an ontology of Government. Next steps will be to use oeGOV to build OWL maps of who is publishing what. The oeGOV ontology files have RDF/OWL or N3 format.

• e-Commerce: The Universal Standard Products and Services Classification Code (UN/SPSC)⁷ is a freely available class taxonomy classifying products and services. Many B2B sites are currently using and extending it to better achieve their particular purposes. The UN/SPSC ontology files have DAML+OIL or RDF format.

• Enterprise Modeling: Ontologies play a major role in this field by creating and maintaining an organizational memory that lets the different enterprise areas interoperate

³http://www.loc.gov/cds/classweb/
⁴http://www.loc.gov/aba/cataloging/subject/weeklylists/
⁤http://www.w3.org/2004/02/skos/
⁶http://www.oegov.org/
⁷http://www.unspsc.org
Figure 1.3: An example of LCSH weekly update in a common language and with unified roles, for example modeling Business Process. They can also be the basis for the agents interoperation language in automated manufacturing processes. The Enterprise Ontology\(^8\) is an example of this kind of ontologies. The formal Ontolingua encoding of the Enterprise Ontology is held in the Library of Ontologies maintained by Stanford University’s Knowledge Systems Lab (KSL).

- Biomedicine/Bioinformatics: In the medical domain, we can find several taxonomies, as Medical Subject Headings (MeSH)\(^9\), which is a comprehensive controlled vocabulary for the purpose of indexing journal articles and books in the life sciences. It can also serve as a thesaurus that facilitates searching. Created by the United States National Library of Medicine (NLM), it is used by the MEDLINE/PubMed article database and by NLM’s catalog of book holdings. MeSH descriptors and qualifiers,

\(^8\)http://www.aiai.ed.ac.uk/project/enterprise/ontology.html
\(^9\)http://www.nlm.nih.gov/mesh/
Figure 1.4: A subsequent weekly update of that shown in Figure 1.3 and Supplementary Concept Records (formerly Supplementary Chemical Records) are in XML format. Files are updated weekly.

Furthermore, there is a growing number of life science ontologies, e.g., the ontologies managed in the OBO (Open Biomedical Ontologies) Foundry [30], which provide a resource where biomedical ontologies are made available in a standard format that allows systematic updating and versioning on the basis of community feedback. Currently, there are nearly 60 ontologies distributed through the OBO website\(^\text{10}\), spanning domains from anatomy (e.g., Mouse adult anatomy) to ethology (Loggerhead nesting), and from gene and gene product features (Sequence Ontology\(^\text{11}\) and Gene Ontology\(^\text{12}\)) to phenotypic qualities knowledge (Disease Ontology\(^\text{13}\)).

The existing ontologies are not static but are frequently evolved to incorporate the

\(^{10}\)http://www.obofoundry.org/

\(^{11}\)http://www.sequenceontology.org/

\(^{12}\)http://www.geneontology.org/

\(^{13}\)http://do-wiki.nubic.northwestern.edu/
newest knowledge of a domain or to adapt to changing application requirements. Just indicatively, there are daily new versions for the popular Gene Ontology. Gene Ontology files are in OBO or RDF/OWL format, while diffs between revisions can be found in “http://cvsweb.geneontology.org/cgi-bin/cvsweb.cgi/go/ontology/”.

- Natural Language Processing: Ontologies can help the semantic analysis of text by representing grammatical structures as related concepts in order to reduce the existent gap in the interpretation of the semantic ambiguity of the natural language. Since then, ontologies can be useful in text mining and machine translation. WordNet\(^{14}\) is an example of such ontologies. WordNet source files are in Prolog. However, they can be represented in RDF/OWL\(^{15}\) format and a change log can be found in “http://www.w3.org/TR/wordnet-rdf/#changelog”.

All the ontologies listed above, evolve over time as they are being updated and new ontology versions are produced. Apart from the expectation of compatible instance migrations between ontology versions, i.e. instance migrations without invalidity problems, there is a need for quality management over the instance descriptions of the new ontology versions. The rising question is how (i.e. based on which rules and principles) the migrated instance descriptions could become as specific as possible according to the new ontology versions. This work contributes in this direction by producing the suggestions that make the instance descriptions as specific as possible and by proposing them to the users (curators) via a specificity lifecycle management process.

1.3 Contribution of this thesis

The contribution of this thesis lies in:

- Formalizing the notion of (possible) specificity.

- Providing principles, rules, and algorithms for computing possibilities after instance migrations in backwards compatible schema evolution case and non-backwards compatible schema evolution case.

\(^{14}\)http://wordnet.princeton.edu/  
\(^{15}\)http://www.w3.org/TR/wordnet-rdf/
• Describing a flexible specificity-aware curation process.

• Presenting a tool named RIMQA for demonstrating the proposed approach.

• Proposing a compact representation for storing the produced possibilities.

• Providing experimental results over real and synthetic datasets.

We could say that from a more general perspective, this thesis contributes in enriching the lifecycle of Semantic Web data with quality management, appropriate for scenarios where ontologies evolve frequently and independently from instance descriptions. As a consequence, this allows adopting iterative and agile ontology modeling approaches, appropriate for open environments like Linked Open Data.

Although we confine ourselves to RDF/S, the results of this work can be applied to any object-oriented conceptual modeling approach that supports classes, inter-class associations, specialization/generalization hierarchies (among classes and among inter-class associations) and instantiation. It could be applied also in object-oriented software engineering, e.g. to aid software upgrade when new versions of software libraries come up.

1.4 Organization of this thesis

Chapter 1 is the introductory chapter of the thesis.

Chapter 2 gives the required background information and notations.

Chapter 3 formalizes the problem using what is called X-partition and provides the fundamentals of our approach.

Chapter 4 describes the transition of X-partitions.

Chapter 5 provides an algorithm for computing the set of possible schema triples of an RDF/S KB, when the current set of schema triples is backwards compatible with the previous one.

Chapter 6 describes the specificity lifecycle management process.

Chapter 7 defines the notion of composite possibilities and provides an algorithm for computing and ranking them.
Chapter 8 provides an algorithm for computing the set of possible instance triples of an RDF/S KB, when the current set of schema triples is not backwards compatible with the previous one.

Chapter 9 describes a prototype system named RIMQA based on the proposed approach, presents a compact representation for possibilities, and provides experimental results.

Chapter 10 discusses how the sequential migrations between ontology versions can lead to a different set of possibilities from the one-step migration from the first to the last ontology version.

Chapter 11 discusses the related work.

Chapter 12 concludes this thesis and identifies issues for further research.

All proofs of Algorithms and Propositions are given in Appendix A. In Appendix B, we provide the list of symbols used in the thesis.
Chapter 2

Background

This chapter introduces notions and notations that shall be used in the sequel. Let \( URI \) be the set of URI references and \( LIT \) be the set of plain and typed literals. In our framework, an RDF/S Knowledge Base (KB) is defined by a set of RDF triples of the form \( (subject \ predicate \ object) \), where \( subject, \ predicate \in URI \) and \( object \in URI \cup LIT \).

Let \( T \) be the set of all possible triples that can be constructed from a countably infinite set of URIs, as well as literals (e.g. strings, integers, float numbers) [11]. Then, an RDF/S KB (for short KB) can be seen as a finite subset \( K \subseteq T \). Apart from the explicitly specified triples of a KB \( K \), other triples can be inferred based on the RDF/S semantics [13]. For this reason, we introduce the notion of closure.

The closure of a KB \( K \), denoted by \( C(K) \), is the set of all triples that either are explicitly asserted or can be inferred from \( K \) based on RDFS-entailment of the RDF/S semantics [13], with the exceptions that (i) we consider in \( C(K) \), extended RDF triples where literals can be subject of triples and (ii) we remove from the RDF and RDFS axiomatic triples [13], the ones that \( rdf:\_i \) terms, for \( i \in \{1,2,\ldots\} \), appear. The first exception is due to the fact that later we define the instances of classes using the formula \( inst_K(c) = \{o \mid (o \ type \ c) \in C(K)\} \) and the instances of classes may contain literals. The second exception is due to the fact that \( rdf:\_i \) terms are infinite and there are not used in our theory.

Essentially, the following derivation rules are used\(^1\):

(i) if \( (c_1 \ subClassOf \ c_2) \) and \( (c_2 \ subClassOf \ c_3) \) then \( (c_1 \ subClassOf \ c_3) \),

\(^1\)The full list of derivation rules, that we consider, is found in the proof of Prop. 10 in Appendix A.
(ii) if \((pr_1 \text{ subPropertyOf } pr_2)\) and \((pr_2 \text{ subPropertyOf } pr_3)\) then \((pr_1 \text{ subPropertyOf } pr_3)\),
(iii) if \((o \text{ type } c_1)\) and \((c_1 \text{ subClassOf } c_2)\) then \((o \text{ type } c_2)\),
(iv) if \((o pr_1 o')\) and \((pr_1 \text{ subPropertyOf } pr_2)\) then \((o pr_2 o')\), and
(v) if \((o pr o')\) then \((o \text{ type } \text{domain}(pr))\) and \((o' \text{ type } \text{range}(pr))\).

**Def. 1** Let \(K\) be a KB. We define the tuple \(\Gamma_K = \langle C_K, Pr_K, \text{domain}, \text{range}, \leq_{cl}^*, \leq_{pr}^* \rangle\),
as follows\(^2\):

- \(C_K\) is the set of classes of \(\mathcal{C}(K)\),
- \(Pr_K\) is the set of properties of \(\mathcal{C}(K)\),
- \(\text{domain}\) is a total function \(\text{domain} : Pr_K \rightarrow C_K\) that maps a property in \(Pr_K\) to its domain,
- \(\text{range}\) is a total function \(\text{range} : Pr_K \rightarrow C_K\) that maps a property in \(Pr_K\) to its range,
- \(\leq_{cl}^*\) is the \text{subClassOf} relation between \(C_K\), and
- \(\leq_{pr}^*\) is the \text{subPropertyOf} relation between \(Pr_K\).

Below we introduce notations for the resources of \(K\), the instances of \(K\), and the instances of a class \(c \in C_K\):

\[
\begin{align*}
\text{Res}_K &= \{o \mid (o \text{ type Resource}) \in \mathcal{C}(K)\} \\
\text{Inst}_K &= \text{Res}_K \setminus (C_K \cup Pr_K) \\
\text{inst}_K(c) &= \{o \mid (o \text{ type } c) \in \mathcal{C}(K)\}
\end{align*}
\]

**Def. 2 (Valid KB)**

We consider a KB \(K\) to be valid if:

(i) the relations \(\leq_{cl}^*\) and \(\leq_{pr}^*\) are acyclic,
(ii) if \(pr \leq_{pr}^* pr'\) then \(\text{domain}(pr) \leq_{cl}^* \text{domain}(pr')\) and \(\text{range}(pr) \leq_{cl}^* \text{range}(pr')\).

**Convention:** In this paper, we consider only valid KBs.

The triples of \(\mathcal{T}\) can be partitioned to schema and instance triples, as shown in Table 2.1 (i.e. the RDF triples that are not schema triples, according to Table 2.1, are instance

\(^2\)Note that according to RDF/S semantics [13], \(\leq_{cl}^*\) and \(\leq_{pr}^*\) are reflexive and transitive relations.
triple | abbreviation
---|---
c type Class | c
c subClassOf c′ | c ≤ₐ c′
pr type Property | pr
pr subPropertyOf pr′ | pr ≤ₐ pr′
pr domain c′′ | domain(pr) = c′′
pr range c′′ | range(pr) = c′′

Table 2.1: Schema and Instance Triples

triples). Instance triples can be further partitioned to class instance triples (having the form (o type c)) and property instance triples (having the form (o pr o′)).
Chapter 3

The Notion of X-partition

This chapter formalizes the problem that we are going to solve and provides the fundamentals of the proposed approach.

Given a KB $K$, below we define the set of cartesian instance triples of $K$.

**Def. 3 (Cartesian Instance Triples)**

Given a KB $K$, the set of cartesian instance triples of $K$, denoted by $B_K^1$, is the union of the class instance triples in $Inst_K \times \{\text{type}\} \times C_K$ and the property instance triples in $Inst_K \times Pr_K \times Inst_K$.

Given a KB $K$, we can distinguish its set of schema triples $S_K$ and its set of instance triples $I_K$, i.e. $K = S_K \cup I_K$. However, for migration purposes, we need to consider only instance triples in $B_K^2$, i.e. those in $I_K \cap B_K$. For notational simplicity we shall hereafter assume that $I_K = I_K \cap B_K$, and we shall use $K = (S_K, I_K)$. We define $C_i(K)$ as the set of explicit and inferred instance triples, specifically $C_i(K) = C(K) \cap B_K$. Clearly, it holds: $I_K \subseteq C_i(K) \subseteq B_K$.

**Def. 4 (Valid Property Instance)**

We shall call a property instance triple $(o \ pr \ o') \in B_K$ valid to add, for short valid, if it satisfies the constraint:

$$(o \in inst_K(\text{domain}(pr))) \land (o' \in inst_K(\text{range}(pr)))$$

(3.1)

---

1 The symbol $B$ in $B_K$ stands for Base.

2 Note that the rest instance triples, e.g. property instances that connect classes, are not interesting for migration purposes, and we ignore them.
Note that if we add to $K$ a property instance triple $(o \text{ pr} o')$ that does not satisfy expression (3.1) of Def. 4, then at least one new class instance triple would be added to $\mathcal{C}_i(K)$, due to derivation rule (v) of RDF/S semantics (see Chapter 2). Furthermore, we should note that several Semantic Web Repositories for well justified reasons do not support derivation rule (v), and consequently do not accept the addition of property instance triples that do not satisfy expression (3.1). To simplify notation, hereafter we shall use $\text{valid}(o, \text{ pr}, o', K)$ to denote expression (3.1). We shall also use $\text{Invalid}(K)$ to refer to the invalid property instance triples of $B_K$, i.e.

$$
\text{Invalid}(K) = \{(o \text{ pr} o') \in B_K \mid \neg \text{valid}(o, \text{ pr}, o', K)\}
$$

Let us also introduce some auxiliary notations. We will define the SubTriples of an instance triple as follows:

$$
\text{SubTriples}((o \text{ type} c)) = \{(o \text{ type} c') \mid c' \leq^* c\}
$$

$$
\text{SubTriples}((o \text{ pr} o')) = \{(o \text{ pr'} o') \mid pr' \leq^*_{\text{pr}} pr\}
$$

If $A$ is a set of instance triples, we define:

$$
\text{SubTriples}(A) = \bigcup_{t \in A} \text{SubTriples}(t)
$$

Given two triples $t$ and $t'$, we shall write: $t \leq t'$ iff $t \in \text{SubTriples}(t')$.

Now, we introduce the notion of X-partition which is fundamental for our work. The main idea is to partition the set of cartesian instance triples $B_K$ into three pairwise disjoint subsets: true, false, and possible instance triples:

- the first comprises $I_K$ and the inferred instance triples (i.e. $\mathcal{C}_i(K)$),
- the second comprises instance triples which are not true (denoted by $M_K$), and
- the last comprises instance triples (denoted by $P_K$) that are possible due to schema evolution.

This is what we will call X-partition.

**Def. 5 (X-partition)**

An X-partition of $B_K$ is a three-fold partition of $B_K$, denoted by $(\mathcal{C}_i(K), M_K, P_K)$, that satisfies the following:
(i) \( C_i(K) = C(K) \cap B_K \).
(ii) \( M_K \) is a lower set wrt \( \leq^3 \), i.e. \( \text{SubTriples}(M_K) = M_K \).
(iii) If an element of \( B_K \) is not valid then it belongs to \( M_K \), i.e. \( \text{Invalid}(K) \subseteq M_K \). □

Note that \( C_i(K) \) is an upper set wrt \( \leq^4 \), as consequence the derivation rules (i) and (ii) of RDF/S semantics (see Chapter 2) and the fact that \( C(K) \) is closed with respect to the closure operator \( C \). Note that (ii) of Def. 5 is reasonable because (a) if \( (o \text{ type } c_1) \in M_K \) and \( c_2 \leq^*_d c_1 \) then it should hold \( (o \text{ type } c_2) \in M_K \) and (b) if \( (o \text{ pr}_1 o') \in M_K \) and \( \text{pr}_2 \leq^*_p \text{pr}_1 \) then it should be \( (o \text{ pr}_2 o') \in M_K \). Additionally, note that from (ii) and the fact that an X-partition is a partition, it follows that it does not exist \( m \in M_K \) and \( p \in P_K \) such that \( p \leq m \). Further, from (ii) (and the fact that an X-partition is a partition), it follows that the triples in \( P_K \) fall into the following two categories:

1. class instance triples in \( (\text{Inst}_K \times \{\text{type}\} \times C_K) \),
2. valid to add property instance triples, i.e. property instance trips whose addition in \( K \) would not add any inferred class instance triple to \( C_i(K) \).

Two useful lemmas follow.

**Lemma 1** \( (P_K \cup C_i(K)) \) is an Upper Set wrt \( \leq \)

1. If \( (o \text{ type } c_2) \in P_K \) and \( c_2 \leq^*_d c_1 \) then \( (o \text{ type } c_1) \in (P_K \cup C_i(K)) \).
2. If \( (o \text{ pr}_2 o') \in P_K \) and \( \text{pr}_2 \leq^*_p \text{pr}_1 \) then \( (o \text{ pr}_1 o') \in (P_K \cup C_i(K)) \). □

**Lemma 2** \( (P_K \) is interval-based wrt \( \leq \)

1. If \( c_1 \leq^*_d c_2 \leq^*_d c_3 \) and \( (o \text{ type } c_1) \), \( (o \text{ type } c_3) \in P_K \) then \( (o \text{ type } c_2) \in P_K \).
2. If \( \text{pr}_1 \leq^*_p \text{pr}_2 \leq^*_p \text{pr}_3 \) and \( (o \text{ pr}_1 o') \), \( (o \text{ pr}_3 o') \in P_K \) then \( (o \text{ pr}_2 o') \in P_K \). □

Lemma 2 says that if \( t \) and \( t' \) belong to \( P_K \) then all \( t_x \) in the interval \([t, t']\) (assuming the \( \leq \) partial order) belong to \( P_K \), too. This lemma can be exploited for specifying compact

---

3 A lower set (else called downward closed set) is a subset \( Y \) of a given partially ordered set \( (X, \leq) \) such that, for all elements \( x \) and \( y \), if \( x \leq y \) and \( y \) is an element of \( Y \), then \( x \) is also in \( Y \).

4 Upper set is the dual notion of lower set.
Let us now discuss how the notion of X-partition can be used in our scenarios.

[MSA case]. Consider a KB $K$ that satisfies the Maximum Specificity Assumption (MSA), i.e. all instances have been described with the most specific classes or properties of the schema that hold in the application domain. We could capture this knowledge by an X-partition in which $P_K = \emptyset$. Since an X-partition is a partition, and $P_K = \emptyset$, it follows that all elements of $B_K$ that are not in $C_i(K)$, should be considered as false (i.e. a form of closed world assumption), and thus it should be $M_K = B_K \setminus C_i(K)$. It is not hard to see that the resulting partition is an X-partition. Obviously (ii) and (iii) of Def. 5 are satisfied, and the X-partition of $B_K$ is:

$$(C_i(K), M_K, P_K) = (C_i(K), B_K \setminus C_i(K), \emptyset)$$

Indeed, it holds that $SubTriples(B_K \setminus C_i(K)) = B_K \setminus C_i(K)$. This is because if $t \in B_K \setminus C_i(K)$ and $t' \leq t$ then $t' \in B_K \setminus C_i(K)$, due to derivation rules (iii) and (iv) of RDF/S semantics (see Chapter 2). Additionally, note that $C_i(K) \cap Invalid(K) = \emptyset$, due to derivation rule (v) of RDF/S semantics. Thus, $Invalid(K) \subseteq B_K \setminus C_i(K)$.

[Open World case]. Now consider the other extreme case, i.e. the case where the MSA does not hold for any instance. For example consider that we start from a KB for which we know nothing regarding its completeness or specificity, and we want to consider that every valid instance triple that can be formed using the ontology and is not certain, it is possible. We can capture this knowledge by an X-partition whose $M_K$ contains only the invalid instance triples of $B_K$, i.e. $M_K = Invalid(K)$. Since an X-partition is a partition of $B_K$, it follows that $P_K = B_K \setminus (C_i(K) \cup M_K) = B_K \setminus (C_i(K) \cup Invalid(K))$. This means that $P_K$ contains all valid instance triples of $B_K$ that do not belong to $C_i(K)$. It is not hard to see that the resulting partition is an X-partition. Again (ii) and (iii) of Def. 5 are satisfied\(^5\) and the X-partition of $B_K$ is:

$$(C_i(K), M_K, P_K) = (C_i(K), Invalid(K), B_K \setminus (C_i(K) \cup Invalid(K)))$$

\(^5\)Statement (ii) is satisfied because if it holds that $\neg valid(o,pr,o',K)$ and $pr' \leq^pr pr$ then certainly it holds that $\neg valid(o,pr',o',K)$, due to (ii) of Def. 2. Thus, $SubTriples(Invalid(K)) = Invalid(K)$. Statement (iii) is obvious.
[Mixed case]. It is not hard to see that we can easily capture various application specific assumptions, e.g. for the case where we know that MSA holds only for a part of the KB. Specifically, assume that the MSA holds only for the class instance triples of a subset $O$ of the instances of $K$, i.e. $O \subseteq \text{Inst}_K$. Then, $M_K = \text{Invalid}(K) \cup \{(o \text{ type } c) \in B_K \setminus C_i(K) \mid o \in O\}$. Analogously, we can express explicitly our assumptions regarding the specificity of property instance triples. In particular, if the MSA holds only for the property instance triples whose subject belongs to a subset $O$ of the instances of $K$ then $M_K = \text{Invalid}(K) \cup \{(o \text{ pr } o') \in B_K \setminus C_i(K) \mid o \in O\}$. If the MSA holds only for the property instance triples whose object belongs to a subset $O$ of the instances of $K$ then $M_K = \text{Invalid}(K) \cup \{(o' \text{ pr } o) \in B_K \setminus C_i(K) \mid o \in O\}$. If the MSA holds only for the property instance triples whose both subject and object belong to a subset $O$ of the instances of $K$ then $M_K = \text{Invalid}(K) \cup \{(o \text{ pr } o') \in B_K \setminus C_i(K) \mid o, o' \in O\}$. Finally, in the case that more than one of the previous conditions hold then $M_K$ is the union of their corresponding “negative statements”. Note that in all cases $\text{SubTriples}(M_K) = M_K$ and $\text{Invalid}(K) \subseteq M_K$. 
Chapter 4

Transition of X-partitions

In this chapter, we describe the transition of X-partitions.

Below, we define the notion of backwards compatibility between two sets of schema triples.

Def. 6 (Backwards Compatibility)
Let \( S \) and \( S' \) be two sets of schema triples. \( S' \) is backwards compatible with \( S \), denoted by \( S \sqsubseteq S' \), iff \( C(S) \subseteq C(S') \).

The following Proposition is used in several of our proofs.

Prop. 1 When a KB \( K = (S_K, I_K) \) evolves to a new KB \( K' = (S_{K'}, I_{K'}) \), where \( I_K = I_{K'} \), it holds that \( \text{Inst}_K = \text{Inst}_{K'} \).

Example 1 Figure 4.1 illustrates two KBs \( K \) and \( K' \), each consisting of schema triples \( (S_K \) and \( S_{K'} \)) and instance triples \( (I_K \) and \( I_{K'} \)). Note that \( S_K \sqsubseteq S_{K'} \). Just indicatively in this example we have: \( \text{domain}(\text{drives}) = \text{Person} \in S'K, \ (\text{Car} \leq^\text{d} \text{Vehicle}) \in S_{K'}, \ \text{while} \ (\text{Car} \leq^*_d \text{Vehicle}) \notin S_K. \) In addition, in this example we have \( I_K = I_{K'} \) and \( \text{Inst}_K = \text{Inst}_{K'} \) (the instances are shown at the bottom of each KB). For example, \( (\text{Bob drives BMW.1}), (\text{Fiat.1 type Car}) \in I_K = I_{K'}, \ \text{while} \ \text{Inst}_K = \text{Inst}_{K'} = \{\text{Fiat.1, BMW.1, Bob, Alice, Computer Science Department, FORTH}\}. \)

Consider that we want to migrate the instance triples of a KB \( K = (S_K, I_K) \), to a schema \( S_{K'} \sqsupseteq S_K \), reaching to a KB \( K' = (S_{K'}, I_K) \). It is not hard to see that it holds \( B_{K'} \sqsupseteq B_K \). Since \( S_{K'} \) is backwards compatible with \( S_K \), every \( b \) that belongs to \( B_K \)
Figure 4.1: An instance migration scenario
Classes and class instances are depicted by ovals. Properties are depicted by rectangles and the letters “d” and “r” are used to denote the domain and the range of a property. Fat arrows denote subClassOf/subPropertyOf relationships, while dashed arrows denote instanceOf relationships.

certainly belongs to $B_{K'}$. The superset relationship between $B_K$ and $B_{K'}$ can be strict (i.e. $B_{K'} \supset B_K$) because $S_{K'}$ can contain new elements (classes or properties) that could be used for generating instance triples to be added to $B_{K'}$.

Prop. 2 If $S_K \subseteq S_{K'}$ then $B_{K'} \supseteq B_K$.

We can check if a set of schema triples $S'$ is backwards compatible with another set of schema triples $S$ by computing the Delta function $\Delta_d(S \rightarrow S')$, defined as follows [39]:

$$\Delta_d(S \rightarrow S') = \{Add(t) \mid t \in S' \setminus C(S)\} \cup \{Del(t) \mid t \in S \setminus C(S')\}$$

Note that $Add(t)$ and $Del(t)$ are not functions on $t$ but strings, where $t$ is replaced by the appropriate schema triple.
Prop. 3 Let $S$ and $S'$ be two sets of schema triples. It holds that: $S \subseteq S'$ iff $\Delta_d(S \rightarrow S')$ contains only add operations.

Suppose that we know the X-partition of $K$, i.e. $(C_i(K), M_K, P_K)$. Our objective is to define the new X-partition, i.e. we want to define the transition:

$$(C_i(K), M_K, P_K) \rightsquigarrow (C_i(K'), M_{K'}, P_{K'})$$

By migrating $I_K$ to $S_{K'}$, we can get $C(K')$, and consequently $C_i(K')$. The rising question is how $M_{K'}$ and $P_{K'}$ are defined. Figure 4.2 illustrates the problem that we are going to solve.

As we mentioned in Chapter 1, we can have conflicts among (a) new positive knowledge inferable from the instance triples and the new schema, (b) new “negative” information inferable from the past negative instance triples and the new schema, and (c) the previously computed possibilities (possible refinements). We resolve such conflicts by considering that (a) has higher priority than (b), and (b) has higher priority than (c). The priorities can be expressed by two postulates.

Def. 7 (X-partition Evolution Postulates)
A transition $(C_i(K), M_K, P_K) \rightsquigarrow (C_i(K'), M_{K'}, P_{K'})$ is consistent if the following postulates are satisfied:

(P1) $C_i(K')$ does not depend on $M_K$ or $P_K$.

Priority of the positive knowledge inferable from the instance triples and the new schema.
(Π2) $M_K \cap P_{K'} = \emptyset$.

Past negative information cannot become possible.

Postulate Π1 gives priority to the new positive knowledge over past negative or possible knowledge. It is consistent with (and reminiscent of) the principle “Recent knowledge prevails the old one” (also, called “Principle of Success” [1] and “Primacy of New Information” [6]).

Postulate Π2 says that past negative information cannot become possible. It follows that past negative information is preserved as long as it does not contradict with the new positive knowledge, as stated by the following proposition.

**Prop. 4 (Inertia Rule for Negatives)**

In the context of a transition $(C_i(K), M_K, P_K) \rightsquigarrow (C_i(K'), M_{K'}, P_{K'})$, it follows that: $M_K \cap P_{K'} = \emptyset$ (Π2) iff $(M_K \setminus C_i(K')) \subseteq M_{K'}$.

Below, we present an example that justifies Postulate Π2.

**Example 2** Consider Figure 4.3. It holds that $I_K = I_{K'} = \{(\text{Whale type Mammal}), (\text{Frog type Amphibian})\}$. Assuming that the MSA holds for KB $K$, it follows that $M_K =$
{(Whale type Amphibian), (Frog type Mammal)}. In $S_{K'}$, two new classes are added Animal and Lay Eggs, as well as the relationships Mammal $\leq_{cl}$ Animal, Amphibian $\leq_{cl}$ Animal, and Lay Eggs $\leq_{cl}$ Animal. Obviously, we want $M_{K'} = M_K$ and $P_{K'} = \{(\text{Whale type Lay Eggs}), (\text{Frog type Lay Eggs})\}$. Note that it holds $M_K \cap P_{K'} = \emptyset$. □

**Notational conventions:** Assume that there exist two KBs $K$ and $K'$. Unless otherwise indicated, with $c$ we will denote a class in both $C_K$ and $C_{K'}$, while with $c'$ we will denote a class in $C_{K'}$ (that possibly belongs also to $C_K$). Analogously, with $pr$ we will denote a property in both $Pr_K$ and $Pr_{K'}$, while with $pr'$ we will denote a property in $Pr_{K'}$ (that possibly belongs also to $Pr_K$). In addition, with $\leq_{cl}$ and $\leq_{pr}$, we will refer to the relations of the new schema $S_{K'}$. Further, we consider that the $\leq_{cl}$ and $\leq_{pr}$ relations, used in defining $\text{SubTriples}(A)$, refer to the relations of the new schema $S_{K'}$.

**Prop. 5 (Derivation of Negatives at a Transition)**

Consider an X-partition $(C_i(K), M_K, P_K)$ based on a schema $S_K$ and suppose we want to define the X-partition after migrating $I_K$ to a backwards compatible schema $S_{K'}$. We can derive $M_{K'}$ using the following rules:

(R1) If $(o \text{ type } c) \in M_K$, $c' \leq_{cl} c$, and $(o \text{ type } c) \notin C_i(K')$ then $(o \text{ type } c') \in M_{K'}$.

(R2) If $(o \text{ pr } o') \in M_K$, $pr' \leq_{pr} pr$, and $(o \text{ pr } o') \notin C_i(K')$, then $(o \text{ pr }' o') \in M_{K'}$.

(R3) If $(o \text{ pr }' o') \in B_{K'}$ and $\neg\text{valid}(o, pr', o', K')$ then $(o \text{ pr }' o') \in M_{K'}$. □

It is not hard to see that any element of $M_{K'}$ that will be derived by the above rules respects the following validity constraints of X-partition (Def. 5): $M_{K'}$ and $C_i(K')$ are disjoint, $M_{K'}$ is a lower set, and $M_{K'}$ contains all invalid property instance triples of $B_{K'}$. Essentially, the above rules produce the following set of instance triples:

$$M_{K'} = \text{Invalid}(K') \cup \text{SubTriples}(M_K \setminus C_i(K'))$$

**Example 3** Consider, for instance, Figure 4.1. Assume that $K$ satisfies the MSA. In $K$, Fiat is not a Person, i.e. $(\text{Fiat type Person}) \in M_K$. In $K'$, we have that $(\text{Fiat type Person}) \notin C_i(K')$. Thus, we have that $(\text{Fiat type Person}) \in M_{K'}$ (from Rule R1). In $K$, Bob is not a Car, i.e. $(\text{Bob type Car}) \in M_K$. In $K'$, a new class is
Algorithm 1 $M_K^\text{Update}(K, M_K, S_K)$

Input: A KB $K$, $M_K$, and a set of schema triples $S_K'$ s.t. $S_K \subseteq S_K'$
Output: $M_K'$

(1) $K' = (S_K', I_K)$;
(2) $M_K' = \{\}$;
/* Part A: Class Instances */
/* Rule R1 */
(3) For all $((o \text{ type } c) \in M_K$ and $(o \text{ type } c) \notin C_i(K'))$ do
(4) For all $(c' \in C_K$ and $c' \leq c)$ do
(5) $M_K' = M_K' \cup \{(o \text{ type } c')\}$;
/* Part B: Property Instances */
/* Rule R2 */
(6) For all $((o \text{ pr o'}) \in M_K$ and $(o \text{ pr o'}) \notin C_i(K'))$ do
(7) For all $(pr' \in Pr_K$, and $pr' \leq pr)$ do
(8) $M_K' = M_K' \cup \{(o \text{ pr o'})\}$;
/* Rule R3 */
(9) For all $((o \text{ pr o'})$ s.t. $o \in \text{Inst}_{K'} \cap URI$, $o' \in \text{Inst}_{K'}$, $pr' \in Pr_K'$, and $\neg valid(o, pr, o', K')$) do
(10) $M_K' = M_K' \cup \{(o \text{ pr o'})\}$;
(11) Return $M_K'$;

inserted that specializes the class $\text{Car}$, called $\text{Van}$. We have that $((\text{Bob type Car}) \notin C_i(K'))$. So, $((\text{Bob type Van}) \in M_K'$ (from Rule R1).

In $K$, Alice does not drive a $\text{BMW}_1$, i.e. $((\text{Alice drives BMW}_1) \in M_K$. In $K'$, we have that $((\text{Alice drives BMW}_1) \notin C_i(K'))$. Thus, we have that $((\text{Alice drives BMW}_1) \in M_K'$ (from Rule R2). In $K$, Alice does not work at FORTH, i.e. $((\text{Alice works at FORTH}) \in M_K$. In $K'$, a new property is inserted that specializes the property works at, called paid from. Alice belongs to the instances of the domain of property paid from and FORTH belongs to the instances of the range of property paid from. We have that $((\text{Alice works at FORTH}) \notin C_i(K'))$. So, $((\text{Alice paid from FORTH}) \in M_K'$ (from Rule R2).

Additionally, note that $((\text{Fiat}_1 \text{ paid from FORTH}) \in M_K'$ (due to Rule R3). This is because $((\text{Fiat}_1 \text{ type Person}) \notin C_i(K')$, and thus it holds $\neg valid(\text{Fiat}_1, \text{paid from}, \text{FORTH}, K')$.

It follows that from $M_K$, $K$, and $S_K'$, we can produce $M_K'$ by Algorithm 1, which is based on Prop. 5.

Now, for getting the X-partition of $B_{K'}$, we define $P_{K'}$ as the set of those instance triples in $B_{K'}$ which are neither in $C_i(K')$ nor in $M_K'$. 

28
Def. 8 (Deriving $P_{K'}$ through $B_{K'}, C_i(K')$ and $M_{K'}$) If we know $C_i(K')$ and $M_{K'}$, then we can derive $P_{K'}$ as follows: $P_{K'} = B_{K'} \setminus (M_{K'} \cup C_i(K'))$. \hfill \textcolor{red}{\square}$

Prop. 6 (Transition Correctness)
The derivation of $M_{K'}$ by the rules of Prop. 5 and of $P_{K'}$ by Def. 8, yields a three-fold partition that is an X-partition (according to Def. 5) and respects postulates II1 and II2 of Def. 7. \hfill \textcolor{red}{\square}$

Note that due to Prop. 5, $M_{K'}$ should contain at least the instance triples in $Invalid(K') \cup \text{SubTriples}(M_K \setminus C_i(K'))$. This is actually the $M_{K'}$ of the X-partition produced by Prop. 6. Therefore, we can say that Prop. 6 produces the X-partition that satisfies postulates II1 and II2 and has the minimum set of negatives.

Regarding the size (number of triples) required for keeping $P_{K'}$ and $M_{K'}$, it is not hard to see that $|P_{K'}|, |M_{K'}| \leq |B_{K'} \setminus C_i(K')|$. Since $|B_{K'}| \leq |Inst_{K'}| * |C_{K'}| + |Inst_{K'}|^2 * |Pr_{K'}|$ and $|I_{K'}| \leq |C_i(K')|$. It follows that in general it holds: $0 \leq |P_{K'}|, |M_{K'}| \leq |Inst_{K'}| * |C_{K'}| + |Inst_{K'}|^2 * |Pr_{K'}| - |I_{K'}|$. 

29
Chapter 5

On Managing X-partitions without

$M$-sets

This chapter provides an algorithm for computing the set of possible instance triples of a KB, when the current set of schema triples is backwards compatible with the previous one. Suppose that we want to migrate the instance triples of a $K = (S_K, I_K)$, to a schema $S_{K'}$ such that $S_K \subseteq S_{K'}$. The question is how we could compute $P_{K'}$.

One approach is to apply the method described in the previous chapter. However, the shortcoming of that approach is that it requires a lot of storage space, since it requires the computation of the whole X-partition and to keep $M_K$ stored. Therefore, below we will investigate whether we can achieve our goal without having to compute $M_{K'}$.

Actually, we prefer an approach requiring only $P_K$. The motivation is that it is reasonable to assume that $|P_K| < |M_K|$, meaning that the $P_K$-based approach for computing $P_{K'}$ requires less space than the $M_K$-based approach (described in the previous chapter). After all, the sought state of a curated KB is a state that satisfies the MSA, and such a state requires keeping only $K$ (since $P_K$ is empty at that case). Instead, the alternative approach would require keeping a non-empty (and possibly very large) $M_K$ even if the MSA holds. In general, the more we approach the MSA, the less storage is required for the $P_K$-based approach, while the opposite holds for the $M_K$-based approach. Also note that the lifecycle management process that we propose in Chapter 6 aims at reducing possibilities and approaching a state satisfying MSA.

Thus, we introduce what we call extended KB.
Def. 9 (Extended KB)

An extended KB, for short eKB, is a pair $\mathcal{E} = (K, P_K)$, where $K$ is a KB and $P_K$ is a set of possible instance triples s.t. $P_K \cap \mathcal{C}(K) = \emptyset$.

Below, we will try to compute $P_{K'}$ just from $K$, $P_K$, and $S_{K'}$. The envisioned process will be correct if it gives the same result ($P_{K'}$) as Def. 8.

In general, $P_{K'}$ can be produced by adding and deleting triples to/from $P_K$, i.e. we can write

$$P_{K'} = (P_K \setminus P_{K,\text{Del}}) \cup P_{K,\text{Add}}$$

where $P_{K,\text{Del}}$ are the elements of $P_K$ that should be deleted from it and $P_{K,\text{Add}}$ are the elements that should be added to $P_K$. Due to priority of new knowledge, we certainly know that

$$P_{K,\text{Del}} \supseteq P_K \cap \mathcal{C}(K')$$

Example 4 Consider Figure 5.1. It holds that $S_K \subseteq S_{K'}$. Note that, in $K$, it holds that $P_K = \{(\text{Toyota Auris type Vehicle})\}$ while, in $K'$, it holds that $(\text{Toyota Auris type Vehicle}) \in \mathcal{C}(K')$. According to Def. 9, it holds that $P_{K'} \cap \mathcal{C}(K') = \emptyset$. So, $P_{K,\text{Del}} = \{(\text{Toyota Auris} \ldots\}$

Figure 5.1: Second Instance Migration Scenario
type Vehicle). Intuitively, \( P_{K,Add} = \{(\text{Toyota Auris type Basic Edition}), (\text{Toyota Auris type Special Edition})\} \).

In fact,

\[ P_{K,Del} = (P_K \cap C_i(K')) \cup (P_K \cap M_{K'}) \]

This is because, the only triples that can be deleted from \( P_K \) belong to \( C_i(K') \) or \( M_{K'} \).

Regarding \( P_{K,Add} \), it should contain instance triples that involve the new classes and properties. The following two Propositions indicate the instance triples to be added to \( P_{K,Add} \).

**Prop. 7** For a new class \( c' \in C_{K'} \setminus C_K \), it holds that: \( (o \ type c') \in P_{K'} \iff \)

(i) \( o \in \text{Inst}_K \cap URI \),

(ii) for all \( c \in C_K \) s.t. \( c' \leq_s c \), it holds that \( (o \ type c) \in (C_i(K') \cup P_K) \), and

(iii) \( (o \ type c') \notin C_i(K') \).

**Prop. 8** For a new property \( pr' \in Pr_{K'} \setminus Pr_K \), it holds that: \( (o \ pr' o') \in P_{K'} \iff \)

(i) \( o \ in URI \) and \( \text{valid}(o, pr', o', K') \),

(ii) for all \( pr \in Pr_K \) s.t. \( pr' \leq_{pr} pr \), it holds that \( (o \ pr o') \in (C_i(K') \cup P_K) \), and

(iii) \( (o \ pr' o') \notin C_i(K') \).

5.1 Algorithmic Perspective

Below we present Algorithm 2, which takes as input a KB \( K \), its set of possible instance triples \( P_K \), and a new set of schema triples \( S_{K'} \) (s.t. \( S_K \sqsubseteq S_{K'} \)). It produces the set of possible instance triples \( P_{K'} \) for the new KB \( K' \), where \( K' = (S_{K'}, I_K) \).

Algorithm 2 is applied to compute the set of possible instance triples of the new KB \( K' = (S_{K'}, I_K) \) based on the original KB \( K \), the original set of possible instance triples \( P_K \), and the new set of schema triples \( S_{K'} \), which is backwards compatible with \( S_K \).

Part A (lines 5-6) follows from Prop. 7 and concerns new classes \( c' \in NC \). If a new class \( c' \) has superclasses \( c \) (which are existing classes in \( K \)), we have to check if \( P_K \) or \( C_i(K') \) includes triples of the form \( (o \ type c) \) for each superclass \( c \), where \( o \in \text{Inst}_K \cap URI \).

Only if this is true and \( (o \ type c') \) does not belong to \( C_i(K') \), we can safely add triples of the form \( (o \ type c') \) to \( P_{K,Add} \). In the case that a new class \( c' \) has no superclasses that are
Algorithm 2 Produce_Possibilities(K, P_K, S_K)

Input: a KB K, its set of possible instance triples P_K, and a set of schema triples S_K', s.t. S_K \subseteq S_K', and K' = (S_K', I_K)

Output: the set of possible instance triples P_K' of the KB K'

1. K' = (S_K', I_K);
2. P_{K, Add} = \emptyset;
3. P_{K, Del} = \emptyset;
4. */ FOR CLASS INSTANCES: */
5. NC = C_K\backslash C_K; /* new classes appearing in K' */
6. /* PART A: New classes */
7. For all (c' \in NC) do /* for each new class */
8. P_{K, Add} = P_{K, Add} \cup \{(o \text{ type } c') \mid o \in \text{Inst}_K \cap \text{URI},
                      \forall c \in C_K \text{ s.t. } c' \leq_o c \text{ it holds that } (o \text{ type } c) \in (C_i(K') \cup P_K),
                      (o \text{ type } c') \notin C_i(K')\};
9. /* PART B: Existing classes */
10. For all (c_1 \in C_K) do /* moving class instance triples from P_K to M_K' due to Rule R1 */
11. P_{K, Del} = P_{K, Del} \cup \{(o \text{ type } c_1) \mid c_2 \in C_K, c_1 \leq_{cl} c_2, and
                      (o \text{ type } c_2) \notin (C_i(K') \cup P_K)\};
12. /* FOR PROPERTIES: */
13. NP = Pr_K \backslash Pr_K'; /* new properties appearing in K' */
14. /* PART C: New properties */
15. For all (pr' \in NP) do /* for each new property */
16. P_{K, Add} = P_{K, Add} \cup \{(o \text{ pr' } o') \mid o \in \text{URI}, valid(o, pr', o', K'),
                      \forall pr \in Pr_K \text{ s.t. } pr' \leq_{pr} pr \text{ it holds that } (o \text{ pr' } o') \in (C_i(K') \cup P_K),
                      (o \text{ pr' } o') \notin C_i(K')\};
17. /* PART D: Existing properties */
18. For all (pr_1 \in Pr_K) do /* moving property instance triples from P_K to M_K, due to Rule R2 */
19. P_{K, Del} = P_{K, Del} \cup \{(o \text{ pr_1 } o') \mid pr_2 \in Pr_K, pr_1 \leq_{pr} pr_2, and
                      pr_1 \leq_{pr} pr_2, and (o \text{ pr_2 } o') \notin C_i(K') \cup P_K)\};
20. P_{K, Del} = P_{K, Del} \cup (P_K \cap C_i(K'));
21. P_{K'} = P_K \backslash P_{K, Del};
22. P_{K'} = P_{K'} \cup P_{K, Add};
23. Return P_{K'};

existing classes in K, we just have to check whether a triple of the form (o \text{ type } c') does not belong to C_i(K') before adding it to P_{K, Add}.

Part B (lines 7-8), concerns existing classes c_1 \in C_K, for which we have to add to P_{K, Del} all those triples (o \text{ type } c_1) \in P_K, if there is a class c_2 \in C_K such that c_1 \leq_{cl} c_2 in K' and (o \text{ type } c_2) \notin (C_i(K') \cup P_K). This is because, since S_K \subseteq S_K', it holds that (o \text{ type } c_2) \notin C_i(K). Thus, (o \text{ type } c_2) \in B_K \backslash (P_K \cup C_i(K)) = M_K. Therefore, it follows from Rule R1 of Prop. 5 that (o \text{ type } c_1) \in M_K'. Note that P_{K, Del} \supseteq P_K \cap M_K'.

Part C (lines 10-11) follows from Prop. 8 and concerns new properties pr' \in NP. If
a new property \( pr' \) has superproperties \( pr \) (which are existing properties in \( K \)), we have to check if \( P_K \) or \( C_i(K') \) includes triples of the form \((o \; pr \; o')\) for each superproperty \( pr \). Only if this is true, it holds that \( valid(o, \; pr', \; o', \; K') \), and \((o \; pr' \; o')\) does not belong to \( C_i(K') \), we can safely add triples of the form \((o \; pr' \; o')\) to \( P_{K,\text{Add}} \). In the case that a new property has no superproperties that are existing properties in \( K \), we just have to check whether it holds \( valid(o, \; pr', \; o', \; K') \) and \((o \; pr' \; o')\) does not belong to \( C_i(K') \) before adding \((o \; pr' \; o')\) to \( P_{K,\text{Add}} \).

Part D (lines 12-13) concerns existing properties \( pr_1 \in Pr_K \), for which we have to add to \( P_{K,\text{Del}} \) all those triples \((o \; pr_1 \; o')\) \( P_K \), if there is a property \( pr_2 \in Pr_K \) such that \( pr_1 \leq_{pr} pr_2 \) in \( K' \) and \((o \; pr_2 \; o')\) \( \notin (C_i(K') \cup P_K) \). This is because, since \( S_K \subseteq S_{K'} \), it holds that \((o \; pr_2 \; o')\) \( \notin C_i(K) \). Thus, \((o \; pr_2 \; o')\) \( \in B_K \setminus (P_K \cup C_i(K)) \). Therefore, it follows from Rule R2 of Prop. 5 that \((o \; pr_1 \; o')\) \( \in M_{K'} \). Note that \( P_{K,\text{Del}} \supseteq P_K \cap M_{K'} \).

In line 14 of Algorithm 2, we add to \( P_{K,\text{Del}} \) the set \( P_K \cap C_i(K') \), because all instance triples that belong to \( C_i(K') \) have to be removed from \( P_K \). At the end (lines 15-16), we have to update \( P_K \) by adding to it the \( P_{K,\text{Add}} \) set and by removing from it the \( P_{K,\text{Del}} \) set. Then, we return \( P_{K'} \). The execution order of the above parts A, B, C, and D does not matter since in any order the output result \( P_{K'} \) is the same.

**Example 5** Consider Figure 4.1 and suppose that \( P_K = \{(\text{Fiat}_1 \; \text{type} \; \text{Vehicle}), \; (\text{Bob} \; \text{uses} \; \text{BMW}_1), \; (\text{Alice} \; \text{works} \; \text{at} \; \text{FORTH})\} \). Executing Algorithm 2, step by step, we have that:

(1) \( K' = (S_{K'}, I_K) \);
(2) \( P_{K,\text{Add}} = \emptyset \);
(3) \( P_{K,\text{Del}} = \emptyset \);
(4) \( NC = \{\text{Van}, \; \text{Jeep}, \; \text{Adult}, \; \text{Institute}, \; \text{University}\} \);
(6) \( P_{K,\text{Add}} = P_{K,\text{Add}} \cup \{(\text{Fiat}_1 \; \text{type} \; \text{Van}), \; (\text{BMW}_1 \; \text{type} \; \text{Van}), \; (\text{Fiat}_1 \; \text{type} \; \text{Jeep}), \; (\text{BMW}_1 \; \text{type} \; \text{Jeep}), \; (\text{Bob} \; \text{type} \; \text{Adult}), \; (\text{Alice} \; \text{type} \; \text{Adult}), \; (\text{Computer Science Department} \; \text{type} \; \text{Institute}), \; (\text{FORTH} \; \text{type} \; \text{Institute}), \; (\text{Computer Science Department} \; \text{type} \; \text{University}), \; (\text{FORTH} \; \text{type} \; \text{University})\} \);
(8) \( P_{K,\text{Del}} = P_{K,\text{Del}} \cup \{\emptyset\} \);
(9) \( NP = \{\text{paid from}\} \);
(11) $P_{K,\text{Add}} = P_{K,\text{Add}} \cup \{(\text{Alice paid from Computer Science Department}), (\text{Bob paid from FORTH})\};$

(13) $P_{K,\text{Del}} = P_{K,\text{Del}} \cup \{(\text{Alice works at FORTH})\};$

Note that, $(\text{Alice related to FORTH}) \notin (P_K \cup C_i(K'))$ and work related to holds in $K'$. So, we have to move $(\text{Alice works at FORTH})$ from $P_K$ to $M_{K'}$ (due to Rule $R2$).

(14) $P_{K,\text{Del}} = P_{K,\text{Del}} \cup \{(\text{Fiat}_1 \text{ type Vehicle}), (\text{Bob uses BMW}_1)\};$

Note that $P_{K,\text{Del}}$ is updated by those triples that belong to $P_K$ and now belong to $C_i(K')$. So, we have to remove them from $P_K$.

(15) $P_{K'} = P_K \setminus P_{K,\text{Del}} =$

$\{(\text{Fiat}_1 \text{ type Vehicle}), (\text{Bob uses BMW}_1), (\text{Alice works at FORTH})\}\$

$\{(\text{Fiat}_1 \text{ type Vehicle}), (\text{Bob uses BMW}_1), (\text{Alice works at FORTH})\} = \emptyset;$

(16) $P_{K'} = \{(\text{Fiat}_1 \text{ type Van}), (\text{BMW}_1 \text{ type Van}), (\text{Fiat}_1 \text{ type Jeep}), (\text{BMW}_1 \text{ type Jeep}), (\text{Bob type Adult}), (\text{Alice type Adult}), (\text{Computer Science Department type Institute}), (\text{FORTH type Institute}), (\text{Computer Science Department type University}), (\text{FORTH type University}), (\text{Alice paid from Computer Science Department}), (\text{Bob paid from FORTH})\};$

(17) Return $P_{K'}$;

In order to explain line 8 in part B of Algorithm 2, consider Figure 4.1. Suppose that we have another version $S_{K''} = S_{K'} \cup \{(\text{Van} \leq_{cl} \text{LoadCarrying Vehicle})\}$, where we have a new specialization relationship, i.e. $\text{Van} \leq_{cl} \text{LoadCarrying Vehicle}$. Then, according to line 8 of Algorithm 2, we have that $P_{K,\text{Del}} = P_{K,\text{Del}} \cup \{(\text{Fiat}_1 \text{ type Van})\}$. This is because it holds that $(\text{Fiat}_1 \text{ type LoadCarrying Vehicle}) \notin P_{K'} \cup C_i(K'')$ and $(\text{Van} \leq_{cl} \text{LoadCarrying Vehicle}) \in S_{K''}$. So, we have to move $(\text{Fiat}_1 \text{ type Van})$ from $P_{K'}$ to $M_{K''}$ (due to Rule $R1$).

Below we present three indicative examples in backwards compatible schema evolution case.

**Example 6** Consider Figure 5.2. It holds that $S_K \subseteq S_{K'}$. Note that, in $K$, $P_K = \emptyset$. In $S_{K'}$, four new classes are added, i.e. $\{\text{Van, Employee, Graduate, Undergraduate}\}$ and also four new specialization relationships, i.e. $\text{Van} \leq_{cl} \text{Car}, \text{Employee} \leq_{cl} \text{Person}, \text{Graduate} \leq_{cl} \text{Student}$ and $\text{Undergraduate} \leq_{cl} \text{Student}$. According to Algorithm 2, in $K'$ it holds that $P_{K'} = \{(\text{John type Employee}), (\text{John type Graduate}), (\text{John type Undergraduate})\}$. □
Example 7 Consider Figure 5.3. It holds that $S_K \subseteq S_{K'}$. Note that, in $K$, $P_K = \emptyset$. In $S_{K'}$, three new classes and one new property are added, i.e. \{Van, Employer, Employee, employs\}, and also four new specialization relationships, i.e. Van $\leq_{cl}$ Car, Employer $\leq_{cl}$ Person, Employee $\leq_{cl}$ Person and employs $\leq_{pr}$ knows. Note that domain(employs) =
Person and range(employs) = Person. According to Algorithm 2, in $K'$ it holds that $P_{K'} = \{(John \text{ type Employer}), (John \text{ type Employee}), (Mary \text{ type Employer}), (Mary \text{ type Employee}), (John \text{ employs Mary})\}$. □

Example 8 Consider Figure 5.4. It holds that $S_K \subseteq S_{K'}$. Note that, in $K$, $P_K = \{(John \text{ knows Alice})\}$. In $S_{K'}$, one new property is added, i.e. \{employs\} and also one new specialization relationship, i.e. employs $\leq_{pr}$ knows. Note that domain(employs) = Person and range(employs) = Person. According to Algorithm 2, in $K'$ it holds that $P_{K'} = \{(John \text{ knows Alice}), (John \text{ employs Mary}), (John \text{ employs Alice})\}$. □

Note that if $(o \text{ type } c) \in C_i(K')$ and $(o \text{ type } c') \notin C_i(K' \cup P_{K'})$, for all $c' \leq_{d}^* c$ and $c' \neq c$, then the MSA property holds for the class instance triple $(o \text{ type } c)$. Similarly, if $(o \text{ pr } o') \in C_i(K')$ and $(o \text{ pr } o') \notin C_i(K' \cup P_{K'})$, for all $pr' \leq_{pr}^* pr$ and $pr' \neq pr$, then the MSA property holds for the property instance triple $(o \text{ pr } o')$.

The following Proposition shows that Algorithm Produce_Possibilities($K, P_K, S_{K'}$) is correct, i.e. that it produces the same $P_{K'}$ as that defined in Def. 8 without the need of computing $B_{K'}$ and $M_{K'}$. 38
Prop. 9 Let $K = (S_K, I_K)$ and let $S_{K'}$ be a set of new schema triples s.t. $S_K \subseteq S_{K'}$ and $K' = (S_{K'}, I_K)$ then $P_{K'} = \text{Produce\_Possibilities}(K, P_K, S_{K'})$. □

Prop. 10 The time complexity of Algorithm 2 is $O(|Inst_K|^2|K'|^2 \star (|K'|^2 + |P_K|))$. □
Chapter 6

Specificity Life Cycle Management

Algorithm 2 showed how schema changes affect the possibilities. In this chapter, we will focus on the management of these possibilities, specifically on operations on data that affect the computed possibilities of the same KB $K$, describing the specificity lifecycle management process.

Suppose a number of instance descriptions that are migrated to a new schema version, and as a consequence our machinery has computed a set of possible instance triples $P_K$. If $o$ is an instance, let us denote by $posTriples(o)$ all those triples that include $o$ and belong to $P_K$. Suppose a system that for an instance $o$ shows to the user a set of possible instance triples $U(o)$ such that $U(o) \subseteq posTriples(o)$. The user then decides whether he/she should add one or more than one of these to the certain knowledge base. After that we should also update $P_K$. Figure 6.1 (on the right) illustrates the proposed process.

In Figure 6.1, on the left, we can see the current migration process, where the curator of a KB downloads its latest ontology version, he migrates the instance descriptions to that version, and then he manually tries to revise some of the migrated descriptions. On the right, we can see the proposed migration process. After the migration of the instance descriptions to the latest ontology version, the system computes the possible instance descriptions by executing either Algorithm 2 (when the new schema is backwards compatible with the previous one) or Algorithm 7 (when the new schema is not backwards compatible with the previous one). Then, an iterative procedure starts, where in each iteration, the curator can select a specific instance and then a ranked subset of its possible descriptions is displayed. The curator can accept or reject some or all of these possible
The Current Migration Process

The curator downloads the latest ontology version

Migration of the metadata to that version

The curator manually may try to revise some of the metadata

Curation Process Completed

The Proposed Migration Process

Activity Initial

The curator downloads the latest ontology version

Migration of the metadata to that version

The curator may try to revise some of the metadata

Activity Final

Instance Selection

Display a ranked subset of possibilities for the selected instance

Accept Possibilities?

[YES] Update KB in Acceptance case (certain and possible part)

[NO] Continue?

[YES] Update KB in No Acceptance case (certain and possible part)

Continued?

[CLOSE]

[CLOSE]

Figure 6.1: Current and Proposed Migration Process

descriptions (of the selected instance) and then the eKB (its certain and possible part) is updated, analogously. Below, we describe formally the updating of the certain and the possible part of the eKB.

What we have to ensure is that the update should: (a) respect the user’s request, (b) reduce uncertainty based on what the user was prompted and decided, and (c) yield a valid eKB (that respects the constraint of Def. 9). To specify exactly the updating we need to introduce notations for the possible class instance triples and property instance triples of an instance o.

\[
\begin{align*}
posTriples^{cl}(o) &= \{ (o \text{ type } c) \mid (o \text{ type } c) \in P_K \} \quad // \text{cl: instance of class} \\
posTriples^{spr}(o) &= \{ (o \text{ pr } o' ) \mid (o \text{ pr } o' ) \in P_K \} \quad // \text{spr: subject of property} \\
posTriples^{opr}(o) &= \{ (o' \text{ pr } o) \mid (o' \text{ pr } o) \in P_K \} \quad // \text{opr: object of property}
\end{align*}
\]
We define the *SupTriples* of an instance triple, as follows:

\[
SupTriples((o \text{ type } c)) = \{(o \text{ type } c') \mid c \leq^* c'\}
\]

\[
SupTriples((o \text{ pr } o')) = \{(o \text{ pr }' o') \mid pr \leq^* pr'\}
\]

Let \(A\) be a set of instance triples. We define\(^1\):

\[
SupTriples(A) = \bigcup_{t \in A} SupTriples(t)
\]

Let \(o\) be an instance. If the system shows to the user all instance triples in \(posTriples^t(o)\), where \(t \in \{pr, spr, opr\}\), and he/she decides that none of these should be added then we should update \(P_K\) as follows\(^2\):

\[
P_K^{up} = P_K \setminus posTriples^t(o)
\]

If the system had prompted to the user only a subset of the possibilities, say \(U(o)\) (where \(U(o) \subseteq posTriples^t(o)\)), and the user had decided that none of these should be added then\(^3\)

\[
P_K^{up} = P_K \setminus SubTriples(U(o))
\]

This ensures that not only \(U(o)\) but also all possibilities which are more “specific” than those in \(U(o)\) will be excluded. Note that the instance triples in \(U(o)\) and thus, the instance triples in \(P_K \cap SubTriples(U(o))\), are actually moved to \(M_K^{up}\).

If the user has selected some of the suggested possibilities, say \(X(o)\), \((X(o) \subseteq U(o))\), to be added, then we should update the certain part of the new eKB \(K^{up} = K \cup X(o)\), as follows \(C(K^{up}) = C(K) \cup SupTriples(X(o))\) (see Prop. 11 below), and then update \(P_K\), accordingly. The latter can be done as follows:

\[
P_K^{up} = P_K \setminus SupTriples(X(o))
\]

\[
P_K^{up} = P_K^{up} \setminus SubTriples(U(o) \setminus SupTriples(X(o)))
\]

The first step excludes from \(P_K\) also the newly entailed instance triples from \(K^{up}\), i.e. \(SupTriples\ (X(o))\), (to satisfy the constraint of Def. 9). The second excludes from \(P_K^{up}\) the instance triples in \(U(o) \setminus SupTriples(X(o))\) and their specializations. Note

---

\(^1\)Note that \(SubTriples(A)\) has been defined in Chapter 3.

\(^2\)In \(P_K^{up}\), \(up\) stands for \(updated\).

\(^3\)Note that except from \(U(o)\), the \(SubTriples(U(o))\) should be removed from \(P_K\).
that the instance triples in $U(o) \setminus SupTriples(X(o))$ and, thus the instance triples in $P_K \cap SubTriples(U(o) \setminus SupTriples(X(o)))$ are actually moved to $M^K_{up}$.

Prop. 11 Let $X \subseteq P_K$. If $K^{up} = K \cup X$ then $C(K^{up}) = C(K) \cup SupTriples(X)$. □

Example 9 Consider the scenario of Figure 6.2. $P_K$ and $posTriples^{cl}(John)$ contain the triples: {
(John type Employee), (John type Manager), (John type Student), (John type Postgraduate), (John type PhD_Student)} (shown enclosed in a dashed frame)⁴. If the system shows to the user all triples in $posTriples^{cl}(John)$ and the user decides that none of these should be added, then $P_K$ should be updated as follows: $P^{up}_K = P_K \setminus posTriples^{cl}(John) = \emptyset$.

Now suppose that the system had showed to the user only three of the five possible instance triples, such as: $U(John) = \{(John type Employee), (John type Student), (John type Postgraduate)\}$. If the user decides that he/she does not want to add any of these triples to the certain part of the $eKB$ then $P_K$ should be updated as follows:

$$P^{up}_K = P_K \setminus SubTriples(\{(John type Employee), (John type Student) (John type Postgraduate)\}) = \emptyset$$

However, if the user decides to add one of these three suggested triples, say the triple $X(John) = \{(John type Postgraduate)\}$ then the $eKB$ has to be updated such that

⁴In the figure, $posCl(John)$ denotes the possible classes of John.
\( K^{up} = K \cup \{(\text{John type Postgraduate})\} \). Note that \( \text{SubTriples}(X(\text{John})) = \{(\text{John type Postgraduate}), (\text{John type PhD Student})\} \) and \( \text{SupTriples}(X(\text{John})) = \{(\text{John type Postgraduate}), (\text{John type Student}), (\text{John type Person})\} \).

The possible part \( P_K \) has to be updated as follows:

\[
P_{K}^{up} = \{(\text{John type Employee}), (\text{John type Manager}), (\text{John type Student}), (\text{John type Postgraduate}) (\text{John type PhD Student})\} \setminus \text{SupTriples}(X(\text{John}))
\]

\[
P_{K}^{up} = \{\text{John type Employee}, (\text{John type Manager}), (\text{John type PhD Student})\}
\]

\[
P_{K}^{up} = P_{K}^{up} \setminus \text{SubTriples}(U(\text{John}) \setminus \text{SupTriples}(X(\text{John})))
\]

\[
P_{K}^{up} = P_{K}^{up} \setminus \text{SubTriples}((\text{John type Employee}))
\]

\[
P_{K}^{up} = ((\text{John type PhD Student}))
\]

\[
\]

The updated certain part of the eKB \( K \), i.e. \( K^{up} \), and the updated \( P_{K}^{up} \) can now be given as input to Algorithm 2, in the case that a new set of schema triples \( S_{K'} \) is available, for generating the new \( P_{K'} \). Note that the set of instance triples that is migrated from \( S_K \) to \( S_{K'} \) is now \( I_{K^{up}} \).

6.1 Ranking Possibilities

In this section, we discuss methods for ranking the possibilities. In general, we can exploit such quantification for various kinds of ranking, e.g. for defining a best-match retrieval model over Semantic Web data, for controlling the amount of possible information that is kept stored, and for aiding the interaction process described previously.

If \( t \) is a triple in \( P_K \) or \( C_i(K) \), we could “score” \( t \) according to a degree of certainty. A naive approach would be:

\[
\text{score}(t) = \begin{cases} 
1 & \text{if } t \in C_i(K) \\
\frac{1}{2} & \text{if } t \in P_K 
\end{cases}
\]
Below we introduce a more sophisticated model for ranking the possible triples. Consider the example of Figure 6.2 and suppose that John was originally classified to the class Person. We can say that Student and Employee are more probable classes than Postgraduate, Manager, and PhD Student. To this end we propose an extension of $P_K$ that we call quantified $P_K$, such that each triple is accompanied by a positive integer that is interpreted as follows: the less this value is the more possible the triple is.

Let $dist_{cl}(c \rightarrow c')$ be the length of the shortest path from class $c$ to class $c'$ comprised from $\leq_{cl}$ relationships (over the reflexive and transitive reduction of $\leq^*_{cl}$). If there is no path from class $c$ to class $c'$ comprised from $\leq_{cl}$ relationships then $dist_{cl}(c \rightarrow c') = \infty$. For example, if $c \leq_{cl} c_1 \leq_{cl} c_2 \leq_{cl} c_3 \leq_{cl} c'$ and there is no shorter path from class $c$ to class $c'$ comprised from $\leq_{cl}$ relationships then $dist_{cl}(c \rightarrow c') = 4$. For each element $(o \text{ type } c)$ in $P_K$, the quantified $P_K$ contains an element $((o \text{ type } c), dist_{Class}(o, c))$, where:

$$dist_{Class}(o, c) = \min\{dist_{cl}(c \rightarrow c') \mid c' \in C_K \text{ and } o \in \text{inst}_K(c')\}$$

So, $dist_{Class}(o, c)$ is the shortest distance of $c$ from one of the certain classes of $o$. In the example of Figure 6.2, the possible class instance triples (John type Student) and (John type Employee) have $dist_{Class}(John, Student) = 1$ and $dist_{Class}(John, Employee) = 1$, respectively. The possible class instance triples (John type Postgraduate) and (John type Manager) have $dist_{Class}(John, Postgraduate) = 2$ and $dist_{Class}(John, Manager) = 2$, respectively. The possible class instance triple (John type PhD Student) has $dist_{Class}(John, PhD Student) = 3$. A measure, similar to $dist_{Class}(o, c)$, appears in [27] for measuring the conceptual distance between two concepts.

Algorithm 3, which is used in order to quantify all possible class instance triples whose subject is an instance $o$, is presented below.

<table>
<thead>
<tr>
<th>Algorithm 3 GetAllDistClass(o, C, dist, K, $P_K$, classRankingMap)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input:</strong> an instance $o$, a set of classes $C$, an integer $dist$, which denotes the distance from the classes in $C$ to one of the certain classes of the instance $o$, a KB $K$, its set of possible instance triples $P_K$ and a map $classRankingMap$, which contains possible class instance triples whose subject is $o$ along with their current ranking value</td>
</tr>
<tr>
<td><strong>Output:</strong> a map $classRankingMap$ which contains possible class instance triples, whose subject is $o$, along with their current ranking value</td>
</tr>
</tbody>
</table>

46
For all \((c \in C)\) do

If \(((o \text{ type } c) \in P_K)\) then

If \((\exists d \text{ s.t. } ((o \text{ type } c), d) \in \text{classRankingMap})\) then

\[
\text{classRankingMap} = \text{classRankingMap} \cup \{((o \text{ type } c), \text{dist})\};
\]

else let \(d \text{ s.t. } ((o \text{ type } c), d) \in \text{classRankingMap}\);

If \((\text{dist} < d)\) then

\[
\text{classRankingMap} = \text{classRankingMap} \setminus \{((o \text{ type } c), d)\};
\]

\[
\text{classRankingMap} = \text{classRankingMap} \cup \{((o \text{ type } c), \text{dist})\};
\]

/*end If*/

/*end If*/

\[
C' = \{c' \in C_K \mid c' \preceq_o c\};
\]

\[
\text{classRankingMap} = \text{GetAllDistClass}(o, C', \text{dist} + 1, K, P_K, \text{classRankingMap});
\]

/*end If*/

/*end For*/

Return \(\text{classRankingMap}\);

At the first call of Algorithm 3, in place of the parameters \(C, \text{dist},\) and \(\text{classRankingMap}\), we put the direct subclasses of the top class, i.e. \(\text{Resource}\), the value 1, and the empty set, respectively. We use the map \(\text{classRankingMap}\) in order to store pairs of the form \((\text{key}, \text{value})\), where the \(\text{key}\) is a possible class instance triple of the form \((o \text{ type } c)\) and the \(\text{value}\) is its current ranking value (i.e. the distance from \(c\) to one of the certain classes of \(o\)). Each time we access a possible class instance triple of the form \((o \text{ type } c)\), we have to check if the key \((o \text{ type } c)\) is contained already in the map. If it holds that, we check if its value is greater than the value of the parameter \(\text{dist}\). If this is true, we have to replace the existing value with the value of the parameter \(\text{dist}\) in the map.

For each possible class \(c\) that we check, we call again Algorithm 3, by replacing the value of the parameter \(C\) with the direct subclasses of \(c\), i.e. \(C'\), and by replacing the
value of the parameter dist with the same one, increased by one unit. For each certain class c that we check, we call again Algorithm 3, by replacing the value of the parameter C with the direct subclasses of c, i.e. C’, and by replacing the value of the parameter dist with the value 1. This is because we want to compute the shortest distance from a possible class of o to one of its certain classes. So, if we meet a certain class of o, we have to make the distance dist to be equal to 1.

As we can see, the Algorithm 3 is executed recursively and thus it makes the procedure faster and more efficient. After the first call, Algorithm 3 returns a map classRankingMap of all possible class instance triples, whose subject is o, along with their ranking value.

Example 10 Consider Figure 6.3. It holds that $S_K \sqsubseteq S_K'$. The new set of classes from K to K’ is {C, D, E, F}. So, according to Algorithm 2, the derived possible class instance triples are: {($o_1$ type C), ($o_1$ type D), ($o_1$ type E), ($o_1$ type F)}.

If we rank the possible class instance triples of the instance $o_1$, using the above formula, i.e. $distClass(o, c)$, where o corresponds to $o_1$ and c corresponds to one of the possible classes of $o_1$, we get the following quantified possible class instance triples: {((($o_1$ type C), 1), (($o_1$ type D), 1), (($o_1$ type E), 2), (($o_1$ type F), 2))}.

For example, note that in the case of the possible class instance triple ($o_1$ type F), there are two paths from F to B (which is a certain class of $o_1$). The first one is $F \leq_{cl} C \leq_{cl} B$ and the second one is $F \leq_{cl} E \leq_{cl} D \leq_{cl} B$. Thus, $dist_{cl}(F \rightarrow B)$ is 2. Additionally, there are two paths from F to A (which is also certain class of $o_1$). The first one is $F \leq_{cl} C \leq_{cl} B \leq_{cl} A$ and the second one is $F \leq_{cl} E \leq_{cl} D \leq_{cl} B \leq_{cl} A$. Thus, $dist_{cl}(F \rightarrow A)$ is 3.
Therefore, \( \text{distClass}(o_1, F) = \min\{2, 3\} = 2. \) 

Similarly, let \( \text{dist}_{pr}(pr \rightarrow pr') \) be the length of the shortest path from property \( pr \) to property \( pr' \) comprised from \( \leq_{pr} \) relationships (over the reflexive and transitive reduction of \( \leq_{pr}^* \), which is unique in our case because \( \leq_{pr} \) is finite and acyclic). If there is no path from property \( pr \) to property \( pr' \) comprised from \( \leq_{pr} \) relationships then \( \text{dist}_{pr}(pr \rightarrow pr') = \infty \). For example, if \( pr \leq_{pr} pr_1 \leq_{pr} pr_2 \leq_{pr} pr_3 \leq_{pr} pr' \) and there is no shorter path from property \( pr \) to property \( pr' \) comprised from \( \leq_{pr} \) relationships then \( \text{dist}_{pr}(pr \rightarrow pr') = 4 \).

We define an auxiliary property \( pr_{\text{dummy}} \) such that if \( pr \in Pr_K \) and there is no \( pr' \in Pr_K \) s.t. \( pr \leq_{pr} pr' \) then we add \( pr \leq_{pr} pr_{\text{dummy}} \). For each element \((o pr o')\) in \( P_K \), the quantified \( P_K \) contains an element \(((o pr o'), \text{distProperty}(o; pr; o'))\), where:

\[
\text{distProperty}(o, pr, o') = \begin{cases} 
\min\{\text{dist}_{pr}(pr \rightarrow pr') \mid pr' \in Pr_K, (o pr' o') \in C(K)\} \\
\text{dist}_{pr}(pr \rightarrow pr_{\text{dummy}}) & \text{otherwise}
\end{cases}
\]

So, \( \text{distProperty}(o, pr, o') \) is the shortest distance of \( pr \) from one of the properties \( pr' \) such that \( (o pr' o') \in C(K) \). In the case that there is no property \( pr' \) such that \( (o pr' o') \in C(K) \) then \( \text{distProperty}(o, pr, o') \) is the distance of \( pr \) from \( pr_{\text{dummy}} \).

The \( \text{distClass}(o) \) and \( \text{distProperty}(o, o') \) values can be used for ranking the possible class instance triples and possible property instance triples, respectively. Such ranking can aid the interaction described earlier. In the example of Figure 6.2, the system will suggest first Student and Employee as possible classes of John. If the user will not select any of these then \( P_K \) will be updated and Postgraduate, Manager, and PhD Student will never be suggested as possible classes (for the instance John).

Algorithm 4, which is used in order to quantify possible property instance triples whose subject is \( o \) and object is \( o' \), is presented below.

\begin{algorithm}
\caption{GetAllDistProperty(o, o', Pr, dist, K, P_K, propRankingMap)}
\begin{algorithmic}
\Require a pair of instances \( o, o' \), a set of properties \( Pr \), an integer \( \text{dist} \), which denotes the distance from the properties in \( Pr \) to one of the certain properties of the instances \( o, o' \), a KB \( K \), its set of possible instance triples \( P_K \) and a map \( \text{propRankingMap} \), which contains possible property instance triples, whose subject is \( o \) and object is \( o' \), along with their current ranking value.
\end{algorithmic}
\end{algorithm}
Output: a map propRankingMap of possible property instance triples, whose subject is o and object is o’, along with their current ranking value.

(1) For all (pr ∈ Pr) do
(2) If ((o pr o’) ∈ PK) then
(3) If (∃ d s.t. ((o pr o’), d) ∈ propRankingMap) then
(4) propRankingMap = propRankingMap ∪ {((o pr o’), dist)};
(5) else let d s.t. ((o pr o’), d) ∈ propRankingMap;
(6) If (dist < d) then
(7) propRankingMap = propRankingMap \ {((o pr o’), d)};
(8) propRankingMap = propRankingMap ∪ {((o pr o’), dist)};
/*end If*/
/*end If*/
(9) Pr’ = {pr’ ∈ PK | pr’ ≤pr pr’};
(10) propRankingMap = GetAllDistProperty(o, o’, Pr’, dist + 1, K, PK, propRankingMap);
/*end If*/
(11) else if (o pr o’) ∈ C(K)) then
(12) Pr’ = {pr’ ∈ PK | pr’ ≤pr pr’};
(13) propRankingMap = GetAllDistProperty(o, o’, Pr’, 1, K, PK, propRankingMap);
/*end If*/
/*end For*/
(14) Return propRankingMap;

At the first call of Algorithm 4, in place of the parameters Pr, dist, and propRankingMap, we put the direct subproperties of the top property, i.e. prdummy, the value 1, and the empty set, respectively. We use the map, i.e. propRankingMap, in order to store pairs of the form (key, value), where the key is a possible property instance triple of the form (o pr o’) and the value is its corresponding ranking value (i.e. the shortest distance from pr to one of the certain properties of o, o’). Each time we access a possible property instance triple of the form (o pr o’), we have to check if the key (o pr o’) is contained already in the map. If it holds that then we check if its value is greater than the value of the parameter dist. If this is true, we have to replace the existing value with the value of the parameter dist
in the map.

For each possible property $pr$ that we check, we call again Algorithm 4, by replacing the value of the parameter $Pr$ with the direct subproperties of $pr$, i.e. $Pr'$, and by replacing the value of the parameter $dist$ with the same one, increased by one unit. For each certain property $pr$ that we check, we call again Algorithm 4, by replacing the value of the parameter $Pr$ with the direct subclasses of $pr$, i.e. $Pr'$, and by replacing the value of the parameter $dist$ with the value 1. This is because we want to compute the shortest distance from a possible property of $o, o'$ to one of their certain properties. So, if we meet a certain property of $o, o'$, we have to make the distance $dist$ to be equal to 1.

As we can see, the Algorithm 4 is executed recursively and thus it makes the procedure faster and more efficient. After the first call, Algorithm 4 returns a map $propRankingMap$ of all possible property instance triples, whose subject is $o$ and object is $o$, along with their ranking value.
Chapter 7

Composite Possibilities

So far $P_K$ contains individual suggestions for a KB $K$. In this chapter, we describe how we can extend $P_K$ and reach a $P_{K}^{\text{ext}}$, called extended set of possibilities, that contains also composite suggestions, where a composite suggestion is a set of instance triples. First, we provide an auxiliary definition.

**Def. 10 (Valid Property Set)**

We define the valid property set of $K$, denoted by $\text{Valid}(K)$, as follows:

$$\text{Valid}(K) = \{(o\ pr\ o') \in B_K \mid \text{valid}(o, pr, o', K) \text{ holds}\}.$$ 

Specifically, $P_{K}^{\text{ext}}$ is not a set of triples like $P_K$, but a family of sets containing:

- one singleton $\{t\}$, for each $t \in P_K$, and
- triple sets of the form $\{h, t\}$ or $\{h_1, t, h_2\}$, where $h, h_1, h_2$ are class instance triples of $P_K$ and $t$ is a property instance triple that (a) it does not belong to $\text{Valid}(K)$, and (b) $t$ would belong to $\text{Valid}(K)$ (thus, can be added to $K$) if $h, h_1, h_2$ were added to $K$.

In other words, the non-singleton elements of $P_{K}^{\text{ext}}$ consist of a hypothesis that is already possible (i.e. $h, h_1, h_2$) and a consequence of that hypothesis (i.e. $t$ can be added to $K$, if $h, h_1, h_2$ were added to $K$).

**Example 11** To grasp the idea, consider the scenario of Figure 7.1, where $(\text{HKR6263 type Car})$ is a possible class instance triple in $K'$. The proposed composite possibility in $K'$ is: $\{(\text{HKR6263 type Car}), (\text{Mary drives HKR6263})\}$. This means that if the user decides to add the class instance triple $(\text{HKR6263 type Car})$ to $K'$ then the property instance triple
(Mary drives HKR6263) can be added to $K'$. However, if the user does not add HKR6263 as an instance of Car, then (Mary drives HKR6263) would not be added to $K'$, since this triple is an element of $\text{Invalid}(K')$, and it belongs to $M_K$. In other words, if the user accepts the proposed composite possibility, both instance triples \{\text{(HKR6263 type Car)}\} and \{\text{(Mary drives HKR6263)}\} will be added to $K'$.

Before we define, the extended set of possibilities of a KB $K$, we provide an auxiliary definition.

\textbf{Def. 11} Let $s, s'$ be two sets of instance triples. We define $s \cong s'$ iff (i) they contain the same property instance triples and (ii) for all class instance triples $t \in s$, it exists $t' \in s'$ s.t. $t \in \text{SubTriples}(t')$.

\textbf{Def. 12 (Extended Set of Possibilities)} Let $K$ be a KB. We define the extended set of possibilities as follows:

\[ P_{\text{ext}}^K = \{ \{p\} \mid p \in P_K \} \cup P_{K}^{\text{comp}}, \]
\[ P_{K}^{\text{comp}} = \text{maximal}_s(\{s \cup \{t\} \mid s \subseteq P_K, t \notin \text{Valid}(K), t \in \text{Valid}(K \cup s), \text{and } t \notin \text{Valid}(K \cup s') \forall s' \subset s\}) \]

Let $\text{Cl}(o) = \{c \mid (o \text{ type } c) \in C_i(K)\}$ and $\text{posCl}(o) = \{c \mid (o \text{ type } c) \in P_K\}$. By taking into account Def. 10, we can reach to the following:
Prop. 12 (Composite possibilities) Let $K$ be a KB. It holds that:

$$P_{K}^{\text{comp}} = P_1 \cup P_2 \cup P_3,$$

where:

$$P_1 = \{ \{h_1, t\} \mid t = (o \text{ pr } o') \notin \text{Valid}(K) \text{ where } o, o' \in \text{Inst}_K, pr \in \text{Pr}_K, \quad h_1 = (o \text{ type } c_1) \in P_K \text{ where } domain(pr) = c_1, range(pr) \in \text{Cl}(o') \}$$

$$P_2 = \{ \{t, h_2\} \mid t = (o \text{ pr } o') \notin \text{Valid}(K) \text{ where } o, o' \in \text{Inst}_K, pr \in \text{Pr}_K, \quad h_2 = (o' \text{ type } c_2) \in P_K \text{ where } range(pr) = c_2, domain(pr) \in \text{Cl}(o) \}$$

$$P_3 = \{ \{h_1, t, h_2\} \mid t = (o \text{ pr } o') \notin \text{Valid}(K) \text{ where } o, o' \in \text{Inst}_K, pr \in \text{Pr}_K, \quad h_1 = (o \text{ type } c_1) \in P_K \text{ where } domain(pr) = c_1, \quad h_2 = (o' \text{ type } c_2) \in P_K \text{ where } range(pr) = c_2 \}$$

It follows that each composite possibility of $P_{K}^{\text{comp}}$ contains either two or three instance triples.

---

**Example 12** The composite possibility of the previous example (Figure 7.1) contains a pair of instance triples. An example with three instance triples is the scenario of Figure 7.2. Suppose that Employer and Employee are new classes in $K'$ and hires is a new property in $K'$ that specializes the property knows in $K$. Suppose $(\text{John knows Mary})$ is a property instance triple included in $K$. In this case, we have in $K'$: $\text{posCl(John)} =$
\{\text{Employer, Employee}\} \text{ and } posCl(Mary) = \{\text{Employer, Employee}\}, \text{ and } P^\text{ext}_K \text{ contains the following composite possibility: } \{(\text{John type Employer}), (\text{John hires Mary}), (\text{Mary type Employee})\}. \text{ Thus, if the user accepts the composite possibility then all instance triples (John type Employer), (Mary type Employee), and (John hires Mary) are added to } K'.

\[\square\]

There is no need to store composite possibilities as they can be computed on demand.

7.1 Ranking Composite Possibilities

In this section, we consider the problem of ranking the composite possibilities in \(P^\text{comp}_K\). Consider the migration shown in Figure 7.3, in which the set of possible instance triples in \(K'\) are the class instance triples that (i) relate Mary with the new subclasses of Person and (ii) relate HKR6263 with the new subclasses of Vehicle. In this scenario, we get the following composite possibilities in \(K'\):

\[s_1: \{(\text{Mary type Adult}), (\text{Mary uses HKR6263}), (\text{HKR6263 type Car})\}\] and

\[s_2: \{(\text{Mary type Millionaire}), (\text{Mary drives HKR6263}), (\text{HKR6263 type Expensive car})\}\]

Consider an element \(s \in P^\text{comp}_K\), where \((o \; pr \; o')\) is the property instance triple that belongs to \(s\). Intuitively, we want the quantification of \(s\) to be low if the certain classes of the ends \((o, o')\) are close to the domain/range of \(pr\) and high if they are not close.

In particular, we define the distance \(d\) of a composite possibility \(s \in P^\text{comp}_K\), as follows:
1. If \( s = \{(o \text{ type } \text{domain}(pr)), (pr \text{ o }'), (o' \text{ type } \text{range}(pr))\} \) then:
\[
d = \text{distClass}(o, \text{domain}(pr)) + \text{distClass}(o', \text{range}(pr))
\]

In other words, \( d \) is the sum of the shortest distance of the domain of \( pr \) from one of the certain classes of \( o \) and the shortest distance of the range of \( pr \) from one of the certain classes of \( o' \).

2. If \( s = \{(o \text{ type } \text{domain}(pr)), (pr \text{ o }')\} \) then \( d = \text{distClass}(o, \text{domain}(pr)) \).

3. If \( s = \{(pr \text{ o }'), (o' \text{ type } \text{range}(pr))\} \) then \( d = \text{distClass}(o', \text{range}(pr)) \).

Note that in cases 2 and 3 above, in order to compute \( d \), we consider only the single possible class instance triple that belongs to \( s \), since it holds \((o' \text{ type } \text{range}(pr)) \in C(K)\) and \((o \text{ type } \text{domain}(pr)) \in C(K)\), respectively.

**Example 13** If we rank the composite possibilities \( s_1 \) and \( s_2 \) derived from Figure 7.3, we get the distances \( d_{s_1} = 2 \) and \( d_{s_2} = 4 \), since \( d_{s_1} = \text{distClass}(\text{Mary, Adult}) + \text{distClass}(\text{HKR6263, Car}) = \text{dist}_{cl}(\text{Adult} \rightarrow \text{Person}) + \text{dist}_{cl}(\text{Car} \rightarrow \text{Vehicle}) = 1 + 1 = 2 \) and \( d_{s_2} = \text{distClass}(\text{Mary, Millionaire}) + \text{distClass}(\text{HKR6263, Expensive Car}) = \text{dist}_{cl}(\text{Millionaire} \rightarrow \text{Person}) + \text{dist}_{cl}(\text{Expensive Car} \rightarrow \text{Vehicle}) = 2 + 2 = 4 \).

So, the composite possibility \( s_1 \), i.e. \( \{\text{(Mary type Adult), (Mary uses HKR6263), (HKR6263 type Car)}\} \), has higher priority than \( s_2 \) and is presented first to the user in the lifecycle management process (see Chapter 6).

Algorithm 6, which is used in order to produce and rank composite possibilities whose property instance triple has subject or object a specific instance \( o \), is presented below. We also present one subroutine of Algorithm 6, i.e. Algorithm 5, which computes the ranking value of a specific possible class instance triple \((o \text{ type } cl)\).

We define the possible instances of a class \( c \in C_K \) as \( \text{posInst}_K(c) = \{o \mid (o \text{ type } c) \in P_K\} \).

---

Algorithm 5 \texttt{GetDistClass}(o, cl, C, dist, K, classRankingMap)

**Input:** an instance \( o \), a class \( cl \) s.t. \((o \text{ type } cl) \in P_K\), a set of classes \( C \), which are superclasses of \( cl \), an integer \( dist \), which denotes the distance from \( cl \) to one of the classes in \( C \), a KB \( K \), and a map \texttt{classRankingMap}, which contains the possible class instance triple \((o \text{ type } cl)\) along with its
current ranking value

Output: a map classRankingMap which contains (o type cl) along with its current ranking value

(1) For all (c ∈ C) do
(2)   If ((o type c) ∈ C_i(K)) then
(3)     If (∃ d s.t. ((o type cl), d) ∈ classRankingMap) then
(4)       classRankingMap = classRankingMap ∪ {((o type cl), dist)};
(5)     else let d s.t. ((o type cl), d) ∈ classRankingMap;
(6)     If (dist < d) then
(7)       classRankingMap = classRankingMap \ {((o type cl), d)};
(8)     classRankingMap = classRankingMap ∪ {((o type cl), dist)};
     /*end If*/
     /*end If*/
     /*end If*/
(9)   C′ = {c′ ∈ C_K | c ≤_cl c′};
(10)  classRankingMap = GetDistClass(o, cl, C′, dist + 1, K, classRankingMap);
     /*end For*/
(11)  Return classRankingMap;

Algorithm 5 ranks a specific possible class instance triple (o type cl). We use the map, i.e. classRankingMap, in order to store a pair of the form (key, value), where the key is (o type cl) and the value is its current ranking value (i.e. the distance from cl to one of the certain classes of o). At the first call of Algorithm 5, C contains the direct superclasses of cl, dist has the value 1, and classRankingMap is the empty set.

For each class c that belongs to the set of classes C, we check if (o type c) belongs to C_i(K). If this is true, we have to check if the key (o type cl), is contained already in the map. If it holds that, we check if the corresponding ranking value is greater than the value of the parameter dist. If this is true, we have to replace the existing value with the value of the parameter dist in the map. Otherwise, we just put the pair (key, value), where the key is (o type cl), and the value is dist, to classRankingMap.

However, if (o type c) does not belong to C_i(K), we call recursively Algorithm 5 (line 10), where C′ corresponds to the direct superclasses of class c and dist corresponds to
the previous value of \(d\text{ist}\), increased by one unit. If we access all classes in \(C\), then we return the \(\text{classRankingMap}\) returned by the recursive calls. As we can see, Algorithm 3 is executed recursively and thus it makes the procedure faster and more efficient.

**Algorithm 6** ProduceAndRankCompPoss\((o, K, P_K)\)

*Input:* an instance \(o\), a KB \(K\), and its set of possible instance triples \(P_K\)

*Output:* a map of composite possibilities along with their ranking value, where the property instance triple of each composite possibility has as subject or object the instance \(o\)

(1) \(\text{totalDist} = 0\);
(2) \(\text{compPoss} = \emptyset\);
(3) \(\text{compPossMap} = \emptyset\);
/* PART A: For all possible classes of \(o\): */
(4) For all \((c \in \text{posCl}(o))\) do
/* SUBPART A.1: For all properties that have as domain a possible class of \(o\), i.e. \(c\): */
(5) For all \((pr \text{ s.t. } \text{domain}(pr) = c)\) do
(6) For all \((o' \in \text{inst}_K(\text{range}(pr)))\) do
(7) \(\text{compPoss} = \{(o \text{ type domain}(pr)), (o \text{ pr } o')\}\);
(8) \(C' = \{c' \in C_K \mid \text{domain}(pr) \leq_{cl} c'\}\);
(9) \(\text{classRankingMap} = \text{GetDistClass}(o, \text{domain}(pr), C', 1, K, \emptyset)\);
(10) Let \(d\) s.t. \(((o \text{ type domain}(pr)), d)\) \(\in\) \(\text{classRankingMap}\);
(11) \(\text{compPossMap} = \text{compPossMap} \cup \{(\text{compPoss}, d)\}\);
/*end For*/
(12) For all \((o' \in \text{posInst}_K(\text{range}(pr)))\) do
(13) \(\text{compPoss} = \{(o \text{ type domain}(pr)), (o \text{ pr } o'), (o' \text{ type range}(pr))\}\);
(14) \(C' = \{c' \in C_K \mid \text{domain}(pr) \leq_{cl} c'\}\);
(15) \(C'' = \{c'' \in C_K \mid \text{range}(pr) \leq_{cl} c''\}\);
(16) \(\text{classRankingMap} = \text{GetDistClass}(o, \text{domain}(pr), C', 1, K, \emptyset)\);
(17) Let \(d_1\) s.t. \(((o \text{ type domain}(pr)), d_1)\) \(\in\) \(\text{classRankingMap}\);
(18) \(\text{classRankingMap} = \text{GetDistClass}(o', \text{range}(pr), C'', 1, K, \emptyset)\);
(19) Let \(d_2\) s.t. \(((o' \text{ type range}(pr)), d_2)\) \(\in\) \(\text{classRankingMap}\);
(20) \(\text{totalDist} = d_1 + d_2\);
(21) \(\text{compPossMap} = \text{compPossMap} \cup \{(\text{compPoss}, \text{totalDist})\}\);
/*end For*/
/*end For*/
/* SUBPART A.2: For all properties that have as range a possible class of \(o\), i.e. \(c\): */
For all \((pr \text{ s.t. } \text{range}(pr) = c)\) do

For all \((o' \in \text{inst}_K(\text{domain}(pr)))\) do

\[
\text{compPoss} = \{(o' \text{ pr } o), (o \text{ type } \text{range}(pr))\};
\]

\[
C' = \{c' \in C_K | \text{range}(pr) \leq c'\};
\]

\[
\text{classRankingMap} = \text{GetDistClass}(o, \text{range}(pr), C', 1, K, \emptyset);
\]

Let \(d \text{ s.t. } ((o \text{ type } \text{range}(pr)), d) \in \text{classRankingMap}\);

\[
\text{compPossMap} = \text{compPossMap} \cup \{(\text{compPoss}, d)\};
\]

/*end For*/

For all \((o' \in \text{posInst}_K(\text{domain}(pr)))\) do

\[
\text{compPoss} = \{(o' \text{ type } \text{domain}(pr)), (o' \text{ pr } o), (o \text{ type } \text{range}(pr))\};
\]

\[
C' = \{c' \in C_K | \text{domain}(pr) \leq c'\};
\]

\[
C'' = \{c'' \in C_K | \text{range}(pr) \leq c''\};
\]

\[
\text{classRankingMap} = \text{GetDistClass}(o', \text{domain}(pr), C', 1, K, \emptyset);
\]

Let \(d_1 \text{ s.t. } ((o' \text{ type } \text{domain}(pr)), d_1) \in \text{classRankingMap}\);

\[
\text{classRankingMap} = \text{GetDistClass}(o, \text{range}(pr), C'', 1, K, \emptyset);
\]

Let \(d_2 \text{ s.t. } ((o \text{ type } \text{range}(pr)), d_2) \in \text{classRankingMap}\);

\[
\text{totalDist} = d_1 + d_2;
\]

\[
\text{compPossMap} = \text{compPossMap} \cup \{(\text{compPoss}, \text{totalDist})\};
\]

/*end For*/

/*end For*/

/*end For*/

/* PART B: For all classes of \(o\): */

For all \((c \in \text{Cl}(o))\)

/* SUBPART B.1: For all properties that have as domain a class of \(o\), i.e. \(c\): */

For all \((pr \text{ s.t. } \text{domain}(pr) = c)\) do

For all \((o' \in \text{posInst}_K(\text{range}(pr)))\) do

\[
\text{compPoss} = \{(o' \text{ pr } o'), (o' \text{ type } \text{range}(pr))\};
\]

\[
C' = \{c' \in C_K | \text{range}(pr) \leq c'\};
\]

\[
\text{classRankingMap} = \text{GetDistClass}(o', \text{range}(pr), C', 1, K, \emptyset);
\]

Let \(d \text{ s.t. } ((o' \text{ type } \text{range}(pr)), d) \in \text{classRankingMap}\);

\[
\text{compPossMap} = \text{compPossMap} \cup \{(\text{compPoss}, d)\};
\]

/*end For*/

/*end For*/

/*end For*/

/* SUBPART B.2: For all properties that have as range a class of \(o\), i.e. \(c\): */

For all \((pr \text{ s.t. } \text{range}(pr) = c)\) do

For all \((o' \in \text{posInst}_K(\text{domain}(pr)))\) do
Algorithm 6 produces and ranks all composite possibilities whose property instance triple has as subject or object a specific instance \( o \). At first (Subpart A.1), for all possible classes of the instance \( o \), i.e. \( c \), we take each property \( pr \) that has as domain the class \( c \). Then, we take all certain and possible instances \( o' \) of the range of \( pr \) and we call Algorithm 5 for ranking each possible class instance triple of the composite possibilities in lines 9 and 16. At the end, we put each composite possibility along with its ranking value in the map \( \text{compPossMap} \).

Afterwards (Subpart A.2), for all possible classes of the instance \( o \), i.e. \( c \), we take each property \( pr \) that has as range the class \( c \). Then, we take all certain and possible instances \( o' \) of the domain of \( pr \) and we call Algorithm 5 for ranking each possible class instance triple of the composite possibilities in lines 26, 33 and 35. At the end, we put each composite possibility along with its ranking value in the map \( \text{compPossMap} \).

Afterwards (Subpart B.1), for all certain classes of \( o \), i.e. \( c \), we take each property \( pr \) that has as range the class \( c \). Then, we take all possible instances \( o' \) of the domain of \( pr \) and we call Algorithm 5 for ranking the possible class instance triple of the composite possibility in line 44. At the end, we put the composite possibility along with its ranking value in the map \( \text{compPossMap} \).

Afterwards (Subpart B.2), for all certain classes of \( o \), i.e. \( c \), we take each property \( pr \) that has as domain the class \( c \). Then, we take all possible instances \( o' \) of the range of \( pr \) and we call Algorithm 5 for ranking the possible class instance triple of the composite
possibility in line 51. At the end, we put the composite possibility along with its ranking value in the map $\text{compPossMap}$. 

Algorithm 6 checks all those cases that produce composite possibilities whose property instance triple has as subject or object an instance $o$ and calls a ranking algorithm (Algorithm 5), which is executed recursively and ranks a specific possible class instance triple.
Chapter 8

Non-backwards Compatible Schema Evolution

This chapter defines and provides an algorithm for computing the set of possible instance triples of a KB, in the non-backwards compatible schema evolution case.

A frequent situation, is the case where the next schema version $S_{K'}$ of a KB $K$ is not backwards compatible with $S_K$, i.e. $S_K \not\subseteq S_{K'}$ (and consequently $C(S_K) \not\subseteq C(S_{K'})$), however the instance triples in $I_K$ are transferred to $K'$, i.e. $K' = (S_{K'}, I_K)$, and $K'$ is a (valid) KB. The elements of $I_K$ refer to (usually leaf) classes and properties of $S_K$ which are preserved in $K'$ automatically by the semantics of RDF/S\(^1\) [13]. The changes between $S_{K'}$ and $S_K$ may include deletions of classes, deletions of properties, changes in the subClassOf/subPropertyOf relations, or changes in the domain and range of properties.

Let $K' = (S_{K'}, I_K)$ be a KB. Obviously, it may hold that $B_K \not\subseteq B_{K'}$. For example, if $(o \text{ type } c) \in B_K$ and $c \in C_K \setminus C_{K'}$ then $(o \text{ type } c) \not\in B_{K'}$.

Our goal is to describe how the rules that are used to derive $M_{K'}$ are modified. Def. 7 defining the postulates $\Pi_1$ and $\Pi_2$, Prop. 5 defining the rules $R1$, $R2$, and $R3$, and Def. 8 defining $P_{K'}$ remain the same. Prop. 4 essentially remains the same, but we have to replace $M_K \setminus C_i(K') \subseteq M_{K'}$ by $(M_K \setminus C_i(K')) \cap B_{K'} \subseteq M_{K'}$. This is because it may exist an instance triple $t \in M_K$ but $t \not\in B_{K'}$, due to deletions of classes and properties in $S_{K'}$.

Now, we add a new postulate $\Pi 3$ that applies to the non-backwards compatible schema evolution case. Postulate $\Pi 3$ expresses that if a triple $t \in B_{K'}$ that existed in $C_i(K)$, does

\[1\] If $(o \text{ type } c) \in K'$ then $(c \text{ type Class}) \in C(K')$ and if $(o \text{ pr } o') \in K'$ then $(pr \text{ type Property}) \in C(K')$. 

63
not exist in $C_i(K')$ then $t$ should go to $M_{K'}$.

**Def. 13 (NBC Evolution Postulates)**

A transition $(C_i(K), M_K, P_K) \rightsquigarrow (C_i(K'), M_{K'}, P_{K'})$ is *consistent* if, apart from the postulates $\Pi_1$ and $\Pi_2$ of Def. 7, the following postulate is satisfied.

(\Pi_3) All elements $t \in B_{K'}$ s.t. $t \in C_i(K) \setminus C_i(K')$ are placed to $M_{K'}$.

Note that postulate \Pi_3 is not needed in the backwards compatible schema evolution case, because in this case it holds that $C_i(K) \subseteq C_i(K')$.

**Example 14** Consider the scenario shown in Figure 8.1, where $S_K \not\sqsubseteq S_{K'}$.

Suppose that $\{(\text{John type Full-time Permanent Employee}), (\text{John type Technical Staff}), (\text{John type Professor})\} \in P_K$. Note that the specialization relationship University Employee $\leq_{cl}$ Permanent Employee, which exists in $S_K$, has been deleted in $S_{K'}$. Note that $(\text{John type Permanent Employee}) \in C_i(K)$, while $(\text{John type Permanent Employee}) \not\in C_i(K')$. According to postulate \Pi_3, $(\text{John type Permanent Employee})$ should go to $M_{K'}$.

The following proposition provides an equivalent form of postulate \Pi_3.

**Prop. 13** In the context of a transition $(C_i(K), M_K, P_K) \rightsquigarrow (C_i(K'), M_{K'}, P_{K'})$, it follows that: $(C_i(K) \setminus C_i(K')) \cap B_{K'} \subseteq M_{K'}$ (\Pi_3) iff $C_i(K) \cap P_{K'} = \emptyset$.

Based on \Pi_3, we provide the following rules that produce (additionally to rules R1, R2, and R3) elements of $M_{K'}$, for the classes and properties that exist in $K'$.

**Prop. 14 (Certain Modification Inheritance Rules)** For class and property instance triples:

(R4) If $(o \ type c) \in C_i(K)$, $c' \leq_{cl}^* c$, and $(o \ type c) \not\in C_i(K')$ then $(o \ type c') \in M_{K'}$.

(R5) If $(o \ pr \ o') \in C_i(K)$, $pr' \leq_{pr}^* pr$, and $(o \ pr \ o') \not\in C_i(K')$ then $(o \ pr' \ o') \in M_{K'}$.

Obviously, any element of $M_{K'}$ that will be derived by the rules R1-R5 respects the following validity constraints of $X$-partition (Def. 5): $M_{K'}$ and $C_i(K')$ are disjoint, $M_{K'}$.

---

\(^2\) NBC stands for non-backwards compatible.

\(^3\) The relationships $\leq_{cl}$ and $\leq_{pr}$ hold in $K'$.  

**Figure 8.1: Certain Modification Inheritance Rules**

is a lower set, and $M_{K'}$ contains all invalid property instance triples of $B_{K'}$. Essentially, rules $R1 - R5$ produce the following set of instance triples:

$$M_{K'} = Invalid(K') \cup SubTriples((M_K \setminus C_i(K')) \cap B_{K'}) \cup SubTriples((C_i(K) \setminus C_i(K')) \cap B_{K'})$$

It is easy to see that the derivation of $M_{K'}$ by the rules $R1$, $R2$, and $R3$ of Prop. 5 and $R4$, $R5$ of Prop. 14, as well as the derivation of $P_{K'}$ by Def. 8, yield an X-partition that respects postulates $\Pi1$ and $\Pi2$ of Def. 7 and $\Pi3$ of Def. 13. This is actually the X-partition that satisfies all postulates $\Pi1$, $\Pi2$, and $\Pi3$ and has the minimum set of negatives.

**Example 15** Continuing Example 14, according to Rule $R4$, (John type Permanent Employee) should go to $M_{K'}$. Further, according to Rule $R4$, (John type Full-time Permanent Employee) must be moved from $P_K$ to $M_{K'}$. \[\square\]

Now, similarly to the case of backwards compatible schema evolution, we will try to
compute $P_{K'}$ from $K$, $P_K$, and $S_{K'}$ without having to know or compute neither $M_K$ nor $M_{K'}$. The envisioned process will be correct if it gives the same result ($P_{K'}$) as Def. 8.

In general, $P_{K'}$ can be produced by adding and deleting triples to/from $P_K$, i.e. we can write $P_{K'} = (P_K \setminus P_{K,del}) \cup P_{K,add}$, where $P_{K,del}$ are the elements of $P_K$ that should be deleted from it and $P_{K,add}$ are the elements that should be added to $P_K$. In fact, $P_{K,del} = (P_K \cap C(K')) \cup (P_K \cap M_{K'}) \cup \{t \in P_K \mid t \notin B_{K'}\}$\(^4\). This is because, the only triples that can be deleted from $P_K$ belong to $C(K') \cup M_{K'}$ or do not belong to $B_{K'}$. Regarding $P_{K,add}$, it should contain instance triples that involve the new classes and properties. The following two Propositions indicate the instance triples to be added to $P_{K,add}$.

**Prop. 15** The same as Prop. 7 but now applies to KBs $K = (S_K, I_K)$ and $K' = (S_{K'}, I_K)$, where $S_K$ and $S_{K'}$ are not necessarily backwards compatible. \(\square\)

**Prop. 16** The same as Prop. 8 but now applies to KBs $K = (S_K, I_K)$ and $K' = (S_{K'}, I_K)$, where $S_K$ and $S_{K'}$ are not necessarily backwards compatible. \(\square\)

Below we present Algorithm 7, which takes as input a KB $K$, its set of possible instance triples $P_K$, and a new set of schema triples $S_{K'}$ s.t. $K' = (S_{K'}, I_K)$ is a KB, and produces the set of possible instance triples $P_{K'}$ for the new KB $K'$. As we can see, the only differences between Algorithm 7 and Algorithm 2 are the new parts C and F (see bold lines).

Note that lines (5-8) of Algorithm 7 and lines (5-8) of Algorithm 2, as well as lines (12-15) of Algorithm 7 and lines (10-13) of Algorithm 2 are the same. However, their proof is different and can be found in the proof of correctness of the respective algorithms (in Appendix A). This is because the second ontology version is not necessarily backwards compatible with the first one and the instance triples $(o \ type c_2)$ or $(o \ pr_2 \ o')$ may belong to $C_i(K)$ and not to $C_i(K')$.

Part C (lines 9-10) concerns class instance triples of the form $(o \ type c)$ that belong to $P_K$ and refer to classes $c$ that have been deleted in $K'$. These class instance triples have to be added to $P_{K,del}$ set, as they do not belong to $B_{K'}$.

\(^4\)Note that $t \notin B_{K'}$ in the case that the class $c$ or property $pr$ appearing in $t$ have been deleted in $K'$. 

66
Algorithm 7 Produce_Possibilities_{NBC}(K, P_K, S_K)
Input: a KB K, its set of possible instance triples P_K, and a new set of schema triples S_K’ s.t. K’ = (S_K’, I_K)
Output: the set of possible instance triples P_K’ of the KB K’

(1) K’ = (S_K’, I_K);
(2) P_{K, Add} = ∅;
(3) P_{K, Del} = ∅;
/* FOR CLASS INSTANCES: */
(4) NC = C_K’ \ C_K; /* new classes appearing in K’ */
/* PART A: New classes */
(5) For all (c’ ∈ NC) do /* for each new class */
(6) P_{K, Add} = P_{K, Add} ∪ \{ (o type c’) \mid o ∈ Inst_K ∩ URI, ∀ c ∈ C_K s.t. c’ ≤_c c it holds that (o type c) ∈ (C_i(K’) ∪ P_K), and (o type c’) /∈ C_i(K’); \}
/* PART B: Existing classes */
(7) For all (c ∈ C_K) do
/* Moving class instance triples from P_K to M_{K’} due to Rule R1 and Rule R4. */
(8) P_{K, Del} = P_{K, Del} ∪ \{ (o type c1) \mid c1 ∈ C_K, c1 ≤_c c2, and (o type c2) /∈ C_i(K’); \}
/* PART C: Removing class instance triples from P_K due to removed classes */
(9) For all ((o type c) ∈ P_K) do
(10) If c /∈ C_K’ then P_{K, Del} = P_{K, Del} ∪ \{ (o type c) \};
/* FOR PROPERTIES: */
(11) NP = Pr_K’ \ Pr_K; /* new properties appearing in K’ */
/* PART D: New properties */
(12) For all (pr’ ∈ NP) do /* for each new property */
(13) P_{K, Add} = P_{K, Add} ∪ \{ (o pr’ o’) \mid o ∈ URI, valid(o, pr’, o’, K’), ∀ pr ∈ Pr_K s.t. pr’ ≤_pr pr it holds that(o pr o’) ∉ (C_i(K’) ∪ P_K), and (o pr’ o’) /∈ C_i(K’); \}
/* PART E: Existing properties */
(14) For all (pr1 ∈ Pr_K) do
/* Moving property instance triples from P_K to M_{K’} due to Rule R2 and R5. */
(15) P_{K, Del} = P_{K, Del} ∪ \{ (o pr1 o’) \mid pr1 ∈ Pr_K, pr1 ≤_pr pr2, and (o pr2 o’) /∈ C_i(K’); \}
/* PART F: Removing property instance triples from P_K due to removed properties, subject out of the domain, or object out of range */
(16) For all ((o pr o’) ∈ P_K) do
(17) If pr /∈ Pr_K’ or ¬valid(o, pr, o’, K’) then
P_{K, Del} = P_{K, Del} ∪ \{ (o pr o’) \};
/* Moving instance triples from P_K to C_i(K’). */
(18) P_{K, Del} = P_{K, Del} ∪ (P_K ∩ C_i(K’));
(19) P’_K = P_K \ P_{K, Del};
(20) P’_K’ = P’_K’ \ P_{K, Add};
(21) Return P’_K’;
Part F (lines 16-17) concerns property instance triples of the form \((o \, pr \, o')\) that belong to \(P_K\) and refer to properties \(pr\) that either have been deleted in \(K'\) or it holds that \(\neg \text{valid}(o, pr, o', K')\), possibly because the domain and/or range of \(pr\) has been changed or specialization relationships between classes that involve (directly or indirectly) the domain and/or range of \(pr\) have been deleted in \(S_{K'}\). These property instance triples have to be added to \(P_{K,\text{Del}}\) set because they either do not belong to \(B_{K'}\) or belong to \(\text{Invalid}(K')\).

In the case that a property, \(pr\), that appears in \(S_K\) and \(I_K\), is removed in \(S_{K'}\) and consequently the statements regarding its domain and range, to restore validity of \(K' = (S_{K'}, I_K)\), we consider that the domain and range of \(pr\), in \(K'\), is the class \(\text{Resource}\). Consider, for instance, Figure 8.2. The property instance triple \((o_1 \, pr_1 \, o_2) \in I_K\) leads to an invalid KB because \(pr_1\) and thus, also its domain and range are deleted in \(K'\). If we consider that the domain and range of \(pr_1\), in \(K'\), is the class \(\text{Resource}\) then the validity of \(K'\) is restored.

**Figure 8.2: Non-Backwards Compatible Schema Evolution**

**Example 16** Consider Figure 8.2. Dashed rectangles denote deleted properties in \(S_{K'}\), which appear in the instance triples of \(K' = (S_{K'}, I_K)\). Note that, even though property \(pr_1\) is removed in \(S_{K'}\), it still exists in \(K'\) as an unconnected element because the instance
triple \((o_1, p_1, o_2) \in I_K\), and thus \((p_1 \text{ type } \text{Property}) \in C(K')\), according to the RDF/S semantics [13]. The domain and range of \(p_1\) is the class \(\text{Resource}\).

Suppose that \(P_K = \{(o_5 \text{ type } F), (o_6 \text{ type } G), (o_6 \text{ type } J), (o_7 \text{ type } C), (o_7 \text{ type } D), (o_7 \text{ type } F), (o_7 \text{ type } G), (o_1 p_2 o_2), (o_5 p_3 o_4)\}\).

Executing Algorithm 7, step by step, we have that\(^5\):

1. \(K' = (S_{K'}, I_K)\);
2. \(P_{K,\text{Add}} = \emptyset\);
3. \(P_{K,\text{Del}} = \emptyset\);
4. \(NC = \{I\}\);
5. \(P_{K,\text{Add}} = P_{K,\text{Add}} \cup \{(o_1 \text{ type } I), (o_3 \text{ type } I), (o_5 \text{ type } I), (o_6 \text{ type } I), (o_7 \text{ type } I)\}\);
6. \(P_{K,\text{Del}} = P_{K,\text{Del}} \cup \{(o_6 \text{ type } G), (o_6 \text{ type } J)\}\);

Note that \(\{(o_6 \text{ type } D), (o_6 \text{ type } A)\} \not\in (P_K \cup C, (K'))\) and \(G \leq_{cl} D, J \leq_{cl} A\) hold in \(K'\). So, \(\{(o_6 \text{ type } G), (o_6 \text{ type } J)\}\) are moved from \(P_K\) to \(M_{K'}\). If fact, \((o_6 \text{ type } G)\) is moved to \(M_{K'}\) due to Rule R1 and \((o_6 \text{ type } J)\) is moved to \(M_{K'}\) due to Rule R4.

7. \(P_{K,\text{Del}} = P_{K,\text{Del}} \cup \{(o_6 \text{ type } F), (o_7 \text{ type } F)\}\);

Note that class \(F\) is removed in \(S_{K'}\).

8. \(NP = \{p_3\}\);

9. \(P_{K,\text{Add}} = P_{K,\text{Add}} \cup \{(o_3 p_3 o_4)\}\);

10. \(P_{K,\text{Del}} = P_{K,\text{Del}} \cup \{(o_5 p_3 o_4)\}\);

Note that \((o_5 p_4 o_4) \not\in (P_K \cup C, (K'))\) and \(p_3 \leq_{pr} p_4\) holds in \(K'\). So, \((o_5 p_3 o_4)\) is moved from \(P_K\) to \(M_{K'}\) (due to Rule R2).

11. \(P_{K,\text{Del}} = P_{K,\text{Del}} \cup \{(o_1 p_2 o_2)\}\);

Note that in \(K'\), \(\text{domain}(p_2) = G\) and \(o_1 \not\in \text{inst}_{K'}(\text{domain}(p_2))\).

12. \(P_{K,\text{Del}} = P_{K,\text{Del}} \cup \{(o_7 \text{ type } C)\}\);

Note that \(P_{K,\text{Del}}\) is updated by those triples that belong to \(P_K\) and now belong to \(C(K')\). So, we have to remove them from \(P_K\).

13. \(P_{K'} = P_K \setminus P_{K,\text{Del}} = \{(o_6 \text{ type } F), (o_6 \text{ type } G), (o_6 \text{ type } J), (o_7 \text{ type } C), (o_7 \text{ type } D), (o_7 \text{ type } F), (o_7 \text{ type } G), (o_1 p_2 o_2), (o_5 p_3 o_4)\} \setminus \)

\(^5\text{Note that } (o_5 \text{ type } D) \in C, (K) \text{ due to derivation rule } (v) \text{ of RDF/S semantics (Chapter 2).}\)
\[(o_6 \text{ type } F), (o_6 \text{ type } G), (o_6 \text{ type } J), (o_7 \text{ type } F), (o_7 \text{ type } C), (o_1 \text{ pr}_2 o_2), (o_5 \text{ pr}_3 o_4)\};

(20) \(P_{K'} = P_K' \cup P_{K,\text{Add}} =\)
\[{(o_7 \text{ type } D), (o_7 \text{ type } G)}\cup{(o_1 \text{ type } I), (o_3 \text{ type } I), (o_5 \text{ type } I), (o_6 \text{ type } I), (o_7 \text{ type } I), (o_3 \text{ pr}_5 o_4)};\]

(21) Return \(P_{K'};\)

We would like to note that Algorithm 7 is more general than Algorithm 2 in the sense that it can be applied even in the backwards compatible schema evolution case. However, Algorithm 2 has less steps and is more efficient.

The following Proposition shows that Algorithm \(\text{Produce\_Possibilities}_{NBC}(K, P_K, S_{K'})\) is correct.

**Prop. 17** Let \(K = (S_K, I_K)\) and let \(S_K\) be the new schema version such that \(K' = (S_{K'}, I_K)\). Then, \(P_{K'} = \text{Produce\_Possibilities}_{NBC}(K, P_K, S_{K'});\)

**Prop. 18** The time complexity of Algorithm 7 is \(O(|\text{Inst}_K|^2 * S^2 * (S^2 + |P_K|))\), where \(S = \max(|K|, |K'|);\)
Chapter 9

Implementation and Application

Issues

This chapter describes a prototype system based on the proposed approach, named RIMQA, proposes a compact representation for possibilities, and provides experimental results.

9.1 Prototype System: Architecture

We have implemented a proof-of-concept prototype, called RIMQA (RDF Instance Migration Quality Assistant)\(^1\), and Figures 9.1-9.9 show some indicative screenshots of the system. The application provides a menu bar, where initially two menus are enabled, i.e. “File” and “Help”. The “File” menu provides the menu items “New Project”, “Open Project”, “Save Project”, “Close Project”, and “Exit”, while the “Help” menu provides a link to the web site of the tool that includes the downloadable tool (.jar file), a manual, and a demo.

The user selects from the menu “File” of the menu bar either (a) to create a new RIMQA project (see Figure 9.1) or (b) to open an existing RIMQA project. If the user selects (a), i.e. to create a new project, a new form becomes visible (see Figure 9.2), where the user gives the project name and selects the source ontology (.rdfs file) and a file that contains instance descriptions (.rdf file) with respect to that ontology. Subsequently, the user

\(^1\)The implementation was based on the RDF Main Memory Model of SWKM (http://139.91.183.30:9090/SWK/\(\text{mainfiles/model.html}\)).
Figure 9.1: RIMQA: Select to create a new RIMQA project

Figure 9.2: RIMQA: Create a new RIMQA project and start the Curation Process
selects the destination ontology (.rdfs file), which is a subsequent version of that ontology and optionally the user selects a file with possible instance descriptions (.rdf file) derived from a previous migration with respect to the source ontology and one of its previous versions. The user must give the namespace URI of the ontology versions (included in .rdfs files). The system then automatically migrates the instance descriptions from the source to the destination ontology. Then, it computes the possible instance triples, according to Algorithm 2 (if the destination ontology is backwards compatible with the source one) or Algorithm 7 (if the destination ontology is not backwards compatible with the source one). After that, if the user presses the “Start Curation” button, the curation process starts. After the instance migration and computation of the new possibilities, the rest menus of the menu bar, i.e. “Statistics” and “Curate”, become enabled. If the user selects the “Statistics” menu, he can see the most indicative statistics about the source and the destination ontology, i.e. (a) the number of original classes, properties, (explicit) schema triples, and instance triples in both ontologies, and (b) the number of added classes and properties, and the number of added and deleted (explicit and inferred) schema triples in the destination ontology. The user can also get information about the possibilities of the source and the destination ontology, e.g. the number of original possible class instance triples and possible property instance triples in both ontologies, and the number of added and deleted possible class instance triples and possible property instance triples in the destination ontology (see Figure 9.3).

To curate the resulting descriptions (“Curate” menu), RIMQA allows the user to select one of the following five choices (see Figure 9.4):

1. **Show All Possible Class Instance Triples** (see Figure 9.5). In this case, all possible class instance triples are listed and the user is able to add (by pressing the “Accept” button) one or more possible class instance triples to the certain part of the extended KB (eKB). Subsequently, the selected possible class instance triples and all their supertriples are added to the certain part of the eKB and they are removed from the multiple choice list and from the possible part of the eKB. The user can also

---

2. Note that we use the RDF format in order to store possibilities, as they are instance triples.
3. For a better graphical representation of the URIs, we replace the namespace URI by its namespace prefix (wherever possible).
4. Recall that eKB = (K, PK) (see Def. 9 in Chapter 5). We refer to K as the certain part of eKB and to PK as the possible part of the eKB.
remove (by pressing the “Reject” button) one or more possible class instance triples from the possible part of the eKB. Subsequently, the selected possible class instance triples and all their subtriples are removed from the multiple choice list and from
the possible part of the eKB. After that, the user selects to save the new certain and possible part of the eKB (by pressing the “Save eKB” button). Note that, in order to create a more functional user interface, we consider that the user can reject a set of possibilities. According to Chapter 6, the explicit rejection of possibilities is not supported, but it is an implicit action, which is derived from the no acceptance of a set of possibilities from the user (recall the update of $P^\text{up}_K$ in the case where the user accepts a subset $X(o)$ from a proposed set of possibilities $U(o)$, i.e. $P^\text{up}_K = P^\text{up}_K \setminus \text{SubTriples}(U(o) \setminus \text{SupTriples}(X(o)))$). In the implemented system, if the user accepts a set of possibilities, the system does not exclude from the possible part of the eKB (i) the possibilities that were not selected and do not belong to the supertriples of the selected possibilities, and (ii) the subtriples of the possibilities in (i). The system just adds to the certain part of the eKB the selected possibilities and their supertriples, and then removes only them from the multiple choice list and from the possible part of the eKB (i.e. $P^\text{up}_K = P_K \setminus \text{SupTriples}(X(o))$). Accordingly, if the user rejects a set of possibilities, the system removes the selected possibilities.
and their subtriples from the multiple choice list and from the possible part of the eKB (i.e. $P_K^{up} = P_K \setminus SubTriples(X(o))$).

2. **Show All Possible Property Instance Triples** (similar graphical interface to the above choice but for possible property instance triples).

3. **Show All Possible Instance Triples** (similar graphical interface to the above choice but for all possible (class and property) instance triples).

4. **Select Instance associated with Possibilities (a) by its URI or (b) by its Possible Classes.**

![Figure 9.6: RIMQA: Select a Possible Instance (by its URI)](image)

If the user selects (a) then a new form becomes visible (see Figure 9.6) and the set of all instances that are associated with possibilities are listed. If the user selects (b) then a new form becomes visible (see Figure 9.7) and the set of all possible classes are listed. The user selects one of them, and if he/she presses “View Possible Class Instances”, then all possible instances associated with the selected class are computed and shown to the user, and the user selects one of them. Then, the buttons “View possible classes of the selected instance”, “View possible property instance triples of the selected instance (as subject)”, “View possible property instance triples of the selected instance (as object)”, and “View composite possibilities of the selected
instance” become enabled. In both cases (a) and (b), the user can select optionally the number of possible class instance triples or possible property instance triples (associated with the selected instance) that wants to be shown. After that, he/she selects one of the following four choices:

- **View possible classes of the selected instance.** If the user presses this button, then a new form is visible (see Figure 9.8). All possible classes of the selected instance are shown ranked to the user in a multiple choice list, where the user can add (by pressing the “Accept” button) one or more possible class instance triples to the certain part of the eKB or the user can remove (by pressing the “Reject” button) one or more possible class instance triples from the possible part of the eKB. If the user selects to add one or more possible classes to the set of certain classes of the selected instance (by pressing the “Accept” button), then the possibilities are recomputed, i.e. the suggestions in the multiple choice list are updated, which means that the selected classes and their superclasses are removed from the multiple choice list and the corresponding possible class

---

5Note that the possible classes, regarding the selected instance, with rank value equal to 1 are shown to the user without regard to the given number.
instance triples are removed from the possible part of the eKB (because they now belong to the certain part of the eKB). The drop-down list of the form which contains all the explicit certain classes of the selected instance is updated analogously. If the user selects to remove one or more possible classes from the set of possible classes of the selected instance (by pressing the “Reject” button), then the possibilities are recomputed, i.e., the suggestions in the multiple choice list are updated, which means that the selected classes and their subclasses are removed from the multiple choice list and the corresponding possible class instance triples are removed from the possible part of the eKB. After that, the user selects to save the new certain and possible part of the eKB (by pressing the “Save eKB” button).

Figure 9.8: RIMQA: View Possible Classes of the selected Instance

- **View possible property instance triples of the selected instance (as subject)** (similar graphical interface to the above choice but for possible property instance triples of the selected instance, as subject).
- **View possible property instance triples of the selected instance (as object)** (similar graphical interface to the above choice but for possible property instance triples of the selected instance, as object).
- **View composite possibilities of the selected instance.** If the user presses this button, then a new form is visible (see Figure 9.9), where the user can add a property instance triple to the certain part of the eKB if and only if he/she accepts one or two possible class instance triples to be added to the certain part of the eKB, as well (as described in Chapter 7). This is achieved by pressing the “Accept” button.

5. **Remove All Possibilities.** The user selects to remove all possibilities (see Figure 9.4).

![Figure 9.9: RIMQA: View Composite Possibilities associated with the selected Instance](image)

Note that in the case where the user selects to save the eKB (by pressing the “Save eKB” button), we store the new instance triples, i.e. the certain part of the eKB, in a .rdf file, called “newCertainModel.rdf” and the new possible instance triples, i.e the possible part of the eKB, in a .rdf file, called “newPossibleModel.rdf”.

Future extensions of the implementation would support a graphical visualization of the suggested possibilities.
9.2 A Compact Representation for Possibilities

We can greatly reduce the size required for the possible instance triples by exploiting various properties that hold. For instance, if two classes, say $c_1$ and $c_2$, are possible classes for an instance $o$ and it holds $c_1 \leq^*_cl c_2$ then all classes between, i.e. all $c'$ such that $c_1 \leq^*_cl c' \leq^*_cl c_2$, are also possible classes for $o$ (recall Lemma 2(1)). This allows devising storage representations based on intervals over the reflexive and transitive reduction of the $\leq^*_cl$ relation. For example, if it holds $c_1 \leq_cl c_2 \leq_cl c_3 \leq_cl \ldots \leq_cl c_{10}$ and all of them are possible classes for $o$ then we can represent them by the interval $[c_1, c_{10}]$.

If for a given instance, $o$, there are several intervals having a common end (it is more probable to have a common right end), then we could save space by adopting a more compact representation, e.g. the intervals $[c_1, c_{10}]$ and $[c_2, c_{10}]$ can be represented by $[[c_1, c_2], c_{10}]$. Moreover, if we have a compact representation of the form $[[c_1, \ldots, c_k], c]$ and $\{c_1, \ldots, c_k\}$ are the leaves of the hierarchy rooted at $c$ then we can even omit their representation and adopt a more declarative method, like $[*, c]$, meaning that all subclasses of $c$ are possible classes for $o$. In case where a class $c$ is a possible class of an instance $o$ but there are not subclasses of $c$ or superclasses of $c$ that are possible classes of $o$ then the corresponding compact representation is $[c, c]$, indicated by a point interval $[c]$, for short.

Accordingly, if two property instance triples, say $(o \text{ pr}_1 o')$ and $(o \text{ pr}_2 o')$, are possible property instance triples for two instances $o$ and $o'$ and it holds $\text{pr}_1 \leq^* \text{pr}_2$ then, for all properties $\text{pr}'$ such that $\text{pr}_1 \leq^*_\text{pr} \text{pr}' \leq^*_\text{pr} \text{pr}_2$, $(o \text{ pr}' o')$ is also a possible property instance triple for $o$ and $o'$ (recall Lemma 2(2)). This allows devising storage representations based on intervals over the reflexive and transitive reduction of the $\leq^*_\text{pr}$ relation. For example, if it holds $\text{pr}_1 \leq_\text{pr} \text{pr}_2 \leq_\text{pr} \text{pr}_3 \leq_\text{pr} \ldots \leq_\text{pr} \text{pr}_{10}$ and $(o \text{ pr}_i o')$, for all $i \in \{1, \ldots, 10\}$, is a possible property instance triple for $o$ and $o'$, then we can represent these possible instance triples by the interval $[\text{pr}_1, \text{pr}_{10}]$. The same storage policy (intervals with the same right end), as in class instance triples above, can be followed in the case of property instance triples.
9.2.1 Data Structure

Figure 9.10 illustrates a data structure for the compact representation, which follows an object-centered storage policy (beneficial for the requirements of the life cycle management) regarding classes. As we can see, on the left there is a list of all the instances, lexicographically ordered. In the middle, there is a list of pointers that each points to an interval on the right (where all the intervals of the compact representation of possibilities are found). Note that every instance $o$ on the left points to a consecutive list of pointers in the middle and thus, to a list of intervals on the right. For example, instances $o_1$ and $o_2$, in Figure 9.10, point to the same intervals, i.e. the first interval $[*, A]$ and the second interval $[(B, C, D), E]$. Figure 9.11 illustrates the corresponding data structure for properties. Note
that two or more pairs of instances may have the same compact representation, so we make those pairs of instances point to the same compact representation. For example, \((o_1, o_2)\) and \((o_2, o_3)\), in Figure 9.11, point to the same intervals, i.e. the first interval \([a]\) and the second interval \([\{b, c, d\}, e]\).

### 9.2.2 Benefits and Shortcomings

The main advantage of adopting the compact representation described above is the space saving that we could achieve.

However, if we execute Algorithms 2 and 7 using the compact representation of possibilities, we can see that the cost of looking for a specific possible instance triple in a compact representation is higher than in an explicit representation of all possibilities. The time complexity of looking for an instance triple in an explicit representation of possibilities is \(O(\log_2(|P_K|))\), if possibilities are lexicographically sorted and binary search is used. The time complexity of looking for a possible instance triple in a compact representation depends on several factors, presented below.

Let \(P_K^{\text{compact}}\) be the compact version of \(P_K\). Let \(P_{\text{compact}}(o)\) be the intervals and \(|P_{\text{compact}}(o)|\) be the number of intervals regarding an instance \(o\). If \(int\) is an interval then we define its degree, denoted by \(\text{degree}(int)\), as the number of classes/properties (other than \(\ast\)) that occur in \(int\), e.g. \(\text{degree}(\{a, b, c\}, d) = 4\). If \(int\) is a point interval then \(\text{degree}(int) = 0\). Let \(t_{\text{search}}(o)\) be the time for locating an instance \(o\) in the list of lexicographically ordered instances (see Figure 9.10). Let \(t_{\text{search}}(o, o')\) be the time for locating a pair of instances \((o, o')\) in the list of lexicographically ordered pairs of instances (see Figure 9.11). Let \(t_{\text{subCheck}}^{\text{cl}}\) be the time for checking a subsumption relationship between classes. Let \(t_{\text{subCheck}}^{\text{pr}}\) be the time for checking a subsumption relationship between properties.

To decide whether a class instance triple \((o \text{ type } c)\) belongs to \(P_K^{\text{compact}}\), requires locating the instance \(o\) and checking every interval which is pointed to by \(o\), i.e. the intervals in \(P_{\text{compact}}(o)\). So, the time complexity of looking for a possible class instance triple is as follows:

\[
\text{Time}((o \text{ type } c) \notin P_K^{\text{compact}}) = t_{\text{search}}(o) + \sum_{i=1}^{\vert P_{\text{compact}}(o) \vert} \text{degree}(int_i) \ast t_{\text{subCheck}}^{\text{cl}}.
\]
where \( P_{\text{compact}}(o) = \{ \text{int}_1, ..., \text{int}_k \} \) and \( k = |P_{\text{compact}}(o)| \).

The above formula calculates the time needed for answering if a specific class instance triple \((o \text{ type } c)\) is contained in the compact representation of possibilities. Suppose that \( P_{\text{compact}}(o) = \{ [\{ c_2, c_3 \}, c_1] \} \). To answer this question, at first, we have to find the instance \( o \) in the lexicographically ordered list of instances. Then, we have to scan all intervals of \( o \) until we find one, say \([\{ c_2, c_3 \}, c_1] \), such that \( c \) is subclass of \( c_1 \) and \( c \) is superclass of at least one of the classes in \( \{ c_2, c_3 \} \).

**Example 17** Consider Figure 9.10. Suppose that we want to find out if the class instance triple \((o_1 \text{ type } K)\) belongs to the compact representation of possibilities. Assume that \( C \leq^*_{cl} K \) and \( K \leq^*_{cl} E \) holds. At first, we find \( o_1 \) from the list of instances and then we check the intervals that \( o_1 \) points to, i.e. \([*, A] \) and \([\{B, C, D\}, E]\). Thus, we first check if \( K \leq^*_{cl} A \) holds. Since this is not true, we check if \( K \leq^*_{cl} E \) and if \((B \leq^*_{cl} K \text{ or } C \leq^*_{cl} K \text{ or } D \leq^*_{cl} K)\). Since these conditions hold, \((o_1 \text{ type } K)\) is a possible class instance triple.

The time complexity \( t_{\text{search}}(o) \) for locating an instance \( o \) in the list of lexicographically ordered instances, if binary search is used, is in \( O(\log_2(|\text{Inst}_K|)) \). The time complexity of class subsumption checking \( t_{\text{cl subCheck}}^1 \), if the DFS or BFS graph traversal algorithms are used (on the graph formed using \( \leq_{cl} \) relationships), is in \( O(|\leq_{cl}|) \). However, given that most RDF triple stores currently use labeling schemes for encoding transitive subsumption relationships, \( t_{\text{cl subCheck}}^1 \) can be even in \( O(1) \) (see [37, 16]).

To decide whether a property instance triple \((o \text{ pr } o')\) belongs to \( P_{\text{compact}}^K \), requires locating the pair of instances \((o, o')\) and checking every interval which is pointed to by \((o, o')\), i.e. the intervals in \( P_{\text{compact}}(o, o') \). So, the time complexity of looking for a possible property instance triple is as follows:

\[
\text{Time}((o \text{ pr } o') \in P_{\text{compact}}^K) = t_{\text{search}}(o, o') + \sum_{i=1}^{|P_{\text{compact}}(o, o')|} \text{degree}(\text{int}_i) \ast t_{\text{pr subCheck}}^1,
\]

where \( P_{\text{compact}}(o, o') = \{ \text{int}_1, ..., \text{int}_k \} \) and \( k = |P_{\text{compact}}(o, o')| \).

The above formula calculates the time needed for answering if a specific property instance triple \((o \text{ pr } o')\) is contained in the compact representation of possibilities. Suppose that \( P_{\text{compact}}(o, o') = \{[\{ pr_2, pr_3 \}, pr_1] \} \). To answer this question, at first, we have to
find the pair of instances \((o, o')\) in the lexicographically ordered list of pairs of instances. Then, we have to scan all intervals of \((o, o')\) until we find one, say \([\{pr_2, pr_3\}, pr_1]\), such that \(pr\) is subproperty of \(pr_1\) and \(pr\) is superproperty of at least one of the properties in \([pr_2, pr_3]\).

**Example 18** Consider Figure 9.11. Suppose that we want to find out if the property instance triple \((o_1 a o_2)\) belongs to the compact representation of possibilities. At first, we find \((o_1, o_2)\) from the list of pairs of instances and check the first interval that \((o_1, o_2)\) points to, i.e. \([a]\). Since the property in this point interval is \(a\), it follows that \((o_1 a o_2)\) is a possible property instance triple. □

The time complexity \(t_{\text{search}}(o, o')\) for locating a pair of instances \((o, o')\) in the list of lexicographically ordered pairs of instances is in \(O(\log_2(|Inst_K|^2))\), if binary search is used. The time complexity of property subsumption checking \(t_{\text{subCheck}}\) is in \(O(| \leq_{pr} |)\), if the DFS or BFS graph traversal algorithms are used. However, if labeling is already in place then \(t_{\text{subCheck}}\) is even in \(O(1)\), as mentioned above.

### 9.3 Experimental Evaluation

We have conducted an experimental evaluation whose objectives is to (a) measure the size in triples of the computed \(P_{K'}\), and (b) measure the time required to compute \(P_{K'}\) for investigating the applicability of this method to large datasets. Regarding datasets and measurements, we adopt the following methodology. For each dataset we get a sequence of schema versions, specifically a sequence of the form \(S_{K_0}, \ldots, S_{K_n}\), where each \(S_{K_i}\) is a set of schema triples, and a set of instance triples \(I_{K}\) w.r.t. the first version of the schema, i.e. \(S_{K_0}\). Subsequently, we migrate \(I_{K}\) to each of the subsequent versions of the schema and for each one we compute the corresponding \(P_{K_i}\). Specifically, we do the migrations \(S_{K_{i-1}} \to S_{K_i}\) for all \(i = 1\ldots n\). We use two scenarios: in the first, we consider that \(P_{K_{i-1}} = \emptyset\) and thus the computation of \(P_{K_i}\) depends only on the current migration, i.e. \(S_{K_{i-1}} \to S_{K_i}\), while in the second \(P_{K_{i-1}}\) has been specified from the migration \(S_{K_{i-2}} \to S_{K_{i-1}}\) (or former migrations).
The implementation is written in Java in the context of SWKM (Semantic Web Knowledge Middleware)\(^6\), and all experiments were carried out in an ordinary laptop with processor Pentium(R) Dual-Core CPU T4200 @2.0 Ghz, 2 GB Ram, running Windows Vista.

One important implementation detail is that we do not have to compute the closure of any of the involved KBs. Instead we check whether a particular triple belongs to the closure and this is done efficiently by exploiting appropriate labeling schemes \(^4\).

Table 9.1 shows the number of classes (i.e. \(|C_i|\)), the new classes added from \(S_K\) to \(S_{K'}\) (i.e. \(|C_i| - |C_{i-1}|\)), the number of properties (i.e. \(|Pr_i|\)), the new properties added from \(S_K\) to \(S_{K'}\) (i.e. \(|Pr_i| - |Pr_{i-1}|\)), the number of explicit instance triples (i.e. \(|I_K|\)) that are migrated and the number of explicit schema triples (i.e. \(|S_K|\)) for each version of the Music Ontology. The last line shows the size difference in schema triples between \(S_K\) and \(S_{K'}\).

\(^6\)http://athena.ics.forth.gr:9090/SWKM

\(^7\)http://www.musicontology.com

<table>
<thead>
<tr>
<th></th>
<th>(v.7)</th>
<th>(v.10)</th>
<th>(v.12)</th>
<th>(v.18)</th>
<th>(v.6)</th>
<th>(v.28)</th>
<th>(v.10.12.2007)</th>
<th>(v.28.10.2008)</th>
<th>(v.13.02.2010)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(</td>
<td>C_i</td>
<td>)</td>
<td>113</td>
<td>94</td>
<td>93</td>
<td>93</td>
<td>94</td>
<td>86</td>
<td>124</td>
</tr>
<tr>
<td>(</td>
<td>C_i</td>
<td>-</td>
<td>C_{i-1}</td>
<td>)</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>(</td>
<td>Pr_i</td>
<td>)</td>
<td>147</td>
<td>167</td>
<td>167</td>
<td>167</td>
<td>174</td>
<td>160</td>
<td>160</td>
</tr>
<tr>
<td>(</td>
<td>Pr_i</td>
<td>-</td>
<td>Pr_{i-1}</td>
<td>)</td>
<td>22</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>14</td>
</tr>
<tr>
<td>(</td>
<td>I_K</td>
<td>)</td>
<td>26</td>
<td>26</td>
<td>26</td>
<td>26</td>
<td>26</td>
<td>26</td>
<td>26</td>
</tr>
<tr>
<td>(</td>
<td>S_K</td>
<td>)</td>
<td>1059</td>
<td>1266</td>
<td>1266</td>
<td>1259</td>
<td>1269</td>
<td>1115</td>
<td>1115</td>
</tr>
<tr>
<td>(</td>
<td>S_i</td>
<td>-</td>
<td>S_{i-1}</td>
<td>)</td>
<td>207</td>
<td>0</td>
<td>-7</td>
<td>10</td>
<td>-154</td>
</tr>
</tbody>
</table>

Table 9.1: \(|C_i|\), \(|Pr_i|\), \(|I_i|\) and \(|S_i|\) for each \(i\) Music Ontology version

\[9.3.1\] Real Data Set

We used the RDF/S versions of Music Ontology\(^7\) which is a formal framework for dealing with music-related information on the Semantic Web, including editorial, cultural and acoustic information. In our experiments, we used the following successive versions: v.7.08.2007, v.10.08.2007, v.12.08.2007, v.18.09.2007, v.6.12.2007, v.28.07.2008, v.28.10.2008 and v.13.02.2010. Since each version is not backwards compatible with the previous ones (although it is migration compatible), Algorithm 7 is used.

Table 9.1 shows the number of classes (i.e. \(|C_i|\)), the new classes added from \(S_K\) to \(S_{K'}\) (i.e. \(|C_i| - |C_{i-1}|\)), the number of properties (i.e. \(|Pr_i|\)), the new properties added from \(S_K\) to \(S_{K'}\) (i.e. \(|Pr_i| - |Pr_{i-1}|\)), the number of explicit instance triples (i.e. \(|I_K|\)) that are migrated and the number of explicit schema triples (i.e. \(|S_K|\)) for each version of the Music Ontology. The last line shows the size difference in schema triples between \(S_K\) and \(S_{K'}\).
As we can see in Table 9.1, version v.18.09.2007 has less schema triples than version v.12.08.2007 and version v.28.07.2008 has less schema triples than version v.6.12.2007. This means that a number of deletions has taken place from v.12.08.2007 to v.18.09.2007 and from v.6.12.2007 to v.28.07.2008, accordingly.

**Number of Possible Triples.** Table 9.2 shows the number of possible triples (i.e. $|P_K'|$) produced during the migration from $S_K$ to $S_{K'}$, assuming that $P_K = \emptyset$, while in Table 9.3 we consider that $P_K \neq \emptyset$ for each migration from $S_K$ to $S_{K'}$. Specifically in Table 9.3, $P_K$ is the one derived by the previous migration. The last lines of these tables show the final $|P_K'|$ for each migration $S_K \rightarrow S_{K'}$. As we can see in Table 9.2, during the migrations v.7.08.2007 $\rightarrow$ v.10.08.2007, v.10.08.2007 $\rightarrow$ v.12.08.2007, and v.12.08.2007 $\rightarrow$ v.18.09.2007, Algorithm 7 does not return any possible triples. Moreover, note that the only possible triples that are produced, are those after the execution of part A of
Algorithm 7, i.e. after the update of $P_K$ as a consequence of the new classes added from $S_K$ to $S_{K'}$. Note that in Table 9.2, only the parts A and D of Algorithm 7 are shown, because the rest parts do not reduce the possible triples, since $P_K = \emptyset$. In Table 9.3, we can see that possible triples are produced again only by part A and we also have deletions by part C of Algorithm 7.

*Execution Times.* Tables 9.4 and 9.5 show the execution times, corresponding to the scenarios of Table 9.2 and 9.3, respectively. We can see that the migration takes just some milliseconds and mainly depends on the number of the new classes (part A) added from $S_K$ to $S_{K'}$.

<table>
<thead>
<tr>
<th>$P_K'$</th>
<th>Execution Times (for migrations where $P_K = \emptyset$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Times</td>
<td>v.7</td>
</tr>
<tr>
<td></td>
<td>.08</td>
</tr>
<tr>
<td>A</td>
<td>11.0</td>
</tr>
<tr>
<td>D</td>
<td>4.7</td>
</tr>
<tr>
<td>TOTAL</td>
<td>15.7</td>
</tr>
</tbody>
</table>

Table 9.4: Execution Times (in msec) for Music Ontology migrations for $|P_K| = 0$

<table>
<thead>
<tr>
<th>$P_K'$</th>
<th>Execution Times (for migrations where $P_K \neq \emptyset$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>for each part of Alg. 7</td>
<td>-&gt;</td>
</tr>
<tr>
<td>$P_K$</td>
<td>v.28.07.2008</td>
</tr>
<tr>
<td>(v.18.09.2007)</td>
<td></td>
</tr>
<tr>
<td>-&gt;</td>
<td>v.28.07.2008</td>
</tr>
<tr>
<td>(v.6.12.2007)</td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>3.5</td>
</tr>
<tr>
<td>B</td>
<td>0.6</td>
</tr>
<tr>
<td>C</td>
<td>0.1</td>
</tr>
<tr>
<td>D</td>
<td>0.8</td>
</tr>
<tr>
<td>E</td>
<td>0.2</td>
</tr>
<tr>
<td>F</td>
<td>0.1</td>
</tr>
<tr>
<td>LINE 18</td>
<td>0.5</td>
</tr>
<tr>
<td>TOTAL</td>
<td>5.8</td>
</tr>
</tbody>
</table>

Table 9.5: Execution Times (in msec) for Music Ontology migrations for $|P_K| \neq 0$
9.3.2 Synthetic Data Set

To conduct experiments over larger datasets and backwards compatible ontologies, we created and used one synthetic data set. Specifically, using the synthetic KB generator described in [34], we created a KB, v.K1, with 100 classes and 300 properties. To obtain a schema whose features resemble those of real ones, the subsumption relation follows a power law distribution. Specifically and accordingly to the metrics used in [35], we set the power-law exponent to 0.5 for the total-degree VR function of the property graph and to 1.7 for the PDF function of the descendants distribution. The depth of the class subsumption hierarchy in the schema is 7. Subsequently, we created three subsequent schemas of v.K1, namely v.K2, v.K3 and v.K4; v.K2 was derived by adding to v.K1 14 new classes as specializations to randomly selected leaf classes of v.K1 and similarly 10 new properties as specializations to randomly selected properties of v.K1; v.K3 was derived by adding to v.K2 14 new classes as specializations to randomly selected leaf classes of v.K2 (where 11 of them existed also in v.K1) and similarly 10 new properties as specializations to randomly selected properties of v.K2 (where 9 of them existed also in v.K1); v.K4 was derived by adding to v.K3 14 new classes as specializations to randomly selected leaf classes of v.K3 (where 13 of them existed also in v.K1) and similarly 10 new properties as specializations to randomly selected properties of v.K3 (where 9 of them existed also in v.K1).

For each class of v.K1 we created 100 instances, while for each property of v.K1 we created 10 property instance triples, among randomly selected instances of the corresponding domain and range classes. Table 9.6 shows the features of these schemas and the number of instance triples.

Number of Possible Triples. Since each version is backwards compatible with the previous ones, Algorithm 2 is used. Table 9.7 shows the number of possible triples (i.e. \(|P_K'|\)) produced during the migration from \(S_K\) to \(S_{K'}\), assuming that \(P_K = \emptyset\), while in Table 9.8 we consider that \(P_K \neq \emptyset\) for each migration from \(S_K\) to \(S_{K'}\). Specifically in Table 9.8, \(P_K\) is the one derived by the previous migration. The last lines of these tables show the

---

8VR stands for Value vs Rank (it measures the relationship between the \(i^{th}\) biggest value and its rank \(i\), assuming a descending order).

9PDF stands for Probability Density Function.
Table 9.6: $|C_i|$, $|Pr_i|$, $|I_i|$ and $|S_i|$ for each $i$ version of the Synthetic data set

<table>
<thead>
<tr>
<th>Versions of Synthetic Data ($S_i$)</th>
<th>v.K1</th>
<th>v.K2</th>
<th>v.K3</th>
<th>v.K4</th>
</tr>
</thead>
<tbody>
<tr>
<td>$</td>
<td>C_i</td>
<td>$</td>
<td>101</td>
<td>115</td>
</tr>
<tr>
<td>$</td>
<td>C_i</td>
<td>-</td>
<td>C_{i-1}</td>
<td>$</td>
</tr>
<tr>
<td>$</td>
<td>Pr_i</td>
<td>$</td>
<td>289</td>
<td>299</td>
</tr>
<tr>
<td>$</td>
<td>Pr_i</td>
<td>-</td>
<td>Pr_{i-1}</td>
<td>$</td>
</tr>
<tr>
<td>$</td>
<td>I_i</td>
<td>$</td>
<td>3790</td>
<td>3790</td>
</tr>
<tr>
<td>$</td>
<td>S_i</td>
<td>$</td>
<td>1166</td>
<td>1219</td>
</tr>
<tr>
<td>$</td>
<td>S_i</td>
<td>-</td>
<td>S_{i-1}</td>
<td>$</td>
</tr>
</tbody>
</table>

Table 9.7: $|P_K'|$ Sizes for Synthetic Data migrations. $|P_K| = 0$

Note that in Table 9.7, migration $v.K2 \rightarrow v.K3$ and $v.K3 \rightarrow v.K4$ produce less possible class instance triples than $v.K1 \rightarrow v.K2$. This is because some of the added classes from $v.K2$ to $v.K3$ (or from $v.K3$ to $v.K4$ respectively) are subclasses of classes added from $v.K1$ to $v.K2$ (or from $v.K1$ to $v.K2$ and from $v.K2$ to $v.K3$ respectively), which have no instances (although the rest classes have instances).

Table 9.8: $|P_K'|$ Sizes for Synthetic Data migrations. $|P_K| \neq 0$

Execution Times. Tables 9.9 and 9.10 show the execution times, corresponding to the scenarios of Table 9.7 and 9.8, respectively. We can observe times that range from 4 seconds to 7 minutes and the cost mainly depends on the number of new classes (part A) and on the number of new properties (part C).
 Execution Times (for migrations where $P_K = \emptyset$)

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1.3</td>
<td>4.8</td>
<td>6.1</td>
</tr>
<tr>
<td>C</td>
<td>4.8</td>
<td>3.7</td>
<td>6.7</td>
</tr>
<tr>
<td>TOTAL</td>
<td>6.1</td>
<td>6.7</td>
<td>4.3</td>
</tr>
</tbody>
</table>

Table 9.9: Execution Times (in sec) for Synthetic Data migrations for $|P_K| = 0$

 Execution Times (for migrations where $P_K \neq \emptyset$)

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>284.7</td>
<td>52.2</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>1.4</td>
<td>0.03</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>111.9</td>
<td>38.3</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>0.3</td>
<td>0.2</td>
<td></td>
</tr>
<tr>
<td>LINE 14</td>
<td>0.04</td>
<td>0.01</td>
<td></td>
</tr>
<tr>
<td>TOTAL</td>
<td>398.5</td>
<td>90.74</td>
<td></td>
</tr>
</tbody>
</table>

Table 9.10: Execution Times (in sec) for Synthetic Data migrations for $|P_K| \neq 0$

Since migration is not an every day task, we can say that the computation of possibilities after a migration takes acceptable time (also take into account that we used an ordinary laptop).

### 9.4 Other Applications

**Software Engineering.** Our approach can be used also in object-oriented software engineering for upgrading libraries. Commonly, custom software relies on several libraries usually bundled in the form of jars. Most libraries evolve over time and their versions in most cases are backwards compatible. A new version of a library usually offers new subclasses of existing classes which provide improved/diversified functioning while respecting the ADT (Abstract Data Type) of the superclass. If the new version of a library is backwards compatible with the previous version, replacing the old version with the new version is enough for upgrading a software that depends on that library. However, this does not allow exploiting the new subclasses of the library: the user has to refer to textual descriptions and release notes in order to identify the new classes/features. An IDE (Integrated Development Environment) could adopt our approach for aiding the developer to upgrade his code. Specifically, it could be used for providing suggestions for refinements for those classes that are used by the code, i.e. those objects that instantiate library
classes, and this can be done gradually by the lifecycle process. We should also mention 
that RDF has been proposed as a data structure for software engineering. As a brief and 
very rough example, and assuming a Java library, each Java class corresponds to an RDF 
class, each public instance variable of a class $A$ with name $v$ and type $B$ corresponds to 
a property $v$ with $domain(v) = A$ and $range(v) = B$, each object $o$ that instantiates a 
class $A$ corresponds to a class instance triple $(o \ type \ A)$, etc.
Chapter 10

Sequences of Transitions

In this chapter, we discuss how the sequential migrations between ontology, i.e. $S_i$, $S_{i+1}$, ..., $S_{n-1}$, $S_n$ can lead to a different set of possibilities from the one-step migration from the first to the last ontology, i.e. from $S_i$ to $S_n$. We introduce how the notion of “negative prejudgement, due to lack of the final (forthcoming) knowledge” affects this different result and we discuss how the user/curator can avoid it.

Let $K_1 = (S_1, I_1)$ be a KB and let $S_2, \ldots, S_n$ be a sequence of new schema versions, resulting to KBs $K_i = (S_i, I_1)$, for $i = 2, \ldots, n$. Now consider the one-step migration of $I_1$ from the first $(S_1)$ to the last $(S_n)$ schema, i.e. consider the transition:

$$(C_1, M_1, P_1) \rightsquigarrow (C_n, M_n, P_n)$$

where $C_1$ is based on $S_1$ and $I_1$, and suppose that $P_1 = \emptyset$.

Now consider a sequential migrations scenario where $I_1$ is migrated to $S_2$, then to $S_3$, and so on, up to $S_n$. So, the scenario consists of the following sequence of $n-1$ transitions:

$$(C_1, M_1, P_1) \rightsquigarrow (C_2', M_2', P_2')$$
$$\rightsquigarrow \ldots$$
$$\rightsquigarrow (C_{n-1}', M_{n-1}', P_{n-1}')$$
$$\rightsquigarrow (C_n', M_n', P_n')$$

Let us now discuss the relationship between the outcome of the one-step migration, i.e. $(C_n, M_n, P_n)$, with respect to the final outcome of the sequential migrations, i.e. with

---

1To keep notations simple, here and below we omit the subscripts $K$ from the notations of $C$, $P$ and $M$.  

\((C'_n, M'_n, P'_n)\). The main points are:

(a) \(C_n = C'_n\), i.e. the certain parts of the resulting partitions are the same. This is due to postulate \(\Pi_1\) of Def. 7.

(b) The rest parts of the partitions (i.e. \(M\) and \(P\)) can be different.

Point (b) can be made evident through a small example, like that of Figure 10.1. The last row (III) shows the one step migration. The first and the second rows show two different sequential migrations that lead to the same final schema. Notice that the first sequence gives the same result with the one step migration, however the second does not.

![Figure 10.1: Different sequences of transitions](image)

[Insight]
We could say that this phenomenon is a kind of “negative prejudgement, due to lack of
the final (forthcoming) knowledge”. In our case, the negative prejudgement is realized by postulate Π2.

**Example 19** (Real World Example) A human could also do the same kind of reasoning. For instance, suppose that A, B, C and o of Figure 10.1 correspond to:
A: Animal
B: Human
C: Give Birth to Live Youngs (for short, Give Birth)
and o corresponds to a person, called Peter. In KB $K_1$, Peter is an instance of Human and assuming that the MSA holds, it follows that $M_1 = \{(Peter \text{ type} \text{ Animal})\}$. Therefore, $P_1 = \emptyset$.

In scenario (I), in KB $K_2$, the relationship Human $\leq_{cl}$ Animal is added. Then, since none of the rules $R1 - R5$ apply, it follows that $P_2 = \{(Peter \text{ type Give Birth})\}$. In KB $K_3$, the relationship Give Birth $\leq_{cl}$ Animal is added. Then again, since none of the rules $R1 - R5$ apply, it follows that $P_2 = \{(Peter \text{ type Give Birth})\}$.

Consider now scenario (II), where in KB $K_2$, the relationship Give Birth $\leq_{cl}$ Animal is added. Then, Rule $R1$ applies and $M_2 = \{(Peter \text{ type Animal}), (Peter \text{ type Give Birth})\}$, while $P_2 = \emptyset$. In KB $K_3$, the relationship Human $\leq_{cl}$ Animal is added. It holds $(Peter \text{ type Animal}) \in C_3$. Additionally, it holds that $M_2 = \{(Peter \text{ type Give Birth})\}$, due to Rule $R1$. Obviously, $P_3 = \emptyset$.

Consider now scenario (III), where in KB $K_3$, the relationships Human $\leq_{cl}$ Animal and Give Birth $\leq_{cl}$ Animal are added. Then, since none of the rules $R1 - R5$ apply, it follows that $P_3 = \{(Peter \text{ type Give Birth})\}$ (as in scenario (I)).

[Suggested Policy]
If no user feedback is expected/given after a migration, then there is no need to compute or store the intermediate $P_i$. Instead, it is better to compute it between the first and the last schema, and only when needed (i.e. just before the curator starts the lifecycle process). In this way, we can bypass the “negative prejudgement” due to lack of the forthcoming knowledge.

Below, we present a proposition that indicates when the one-step migration gives the same result with the sequential migrations scenario.
**Prop. 19** Let \( K_1 = (S_1, I_1) \) be a KB and let \( S_2, \ldots, S_n \) be a sequence of backwards compatible schema versions, resulting to KBs \( K_i = (S_i, I_i) \), for \( i = 2, \ldots, n \). Now consider the one-step migration of \( I_1 \) from the first \((S_1)\) to the last \((S_n)\) schema: \((C_1, M_1, P_1) \rightsquigarrow (C_n, M_n, P_n)\). Additionally, consider the sequential migrations scenario, where: \((C_1, M_1, P_1) \rightsquigarrow (C_2', M_2', P_2') \rightsquigarrow \ldots \rightsquigarrow (C_n', M_n', P_n')\). If (a) for all \( c_1, c_2 \in C_{K_1} \), it holds that \( c_1 \leq_{cl} c_2 \) in \( K_1 \) iff \( c_1 \leq_{cl} c_2 \) in \( K_n \), and (b) for all \( pr_1, pr_2 \in Pr_{K_1} \), it holds that \( pr_1 \leq_{pr} pr_2 \) in \( K_1 \) iff \( pr_1 \leq_{pr} pr_2 \) in \( K_n \), then \((C_n, M_n, P_n) = (C_n', M_n', P_n')\).

We would like to note that Prop. 19 does not hold in the case that the sequence of schema versions is not backwards compatible (see Example 10.2). This happens because a class or property that exist in a schema version \( S_i \), they may have been removed from the next schema version \( S_{i+1} \).

![Figure 10.2: Different sequences of non-backwards compatible transitions](image)

**Example 20** Consider Figure 10.2. In KB \( K_1 \), there exist two classes \( A \) and \( B \) and \( o \) is an instance of \( A \). Assuming that the MSA holds for \( K_1 \), it follows that \( P_1 = \emptyset \) and \( M_1 = \{ (o \text{ type } B) \} \). In scenario (I), in KB \( K_2 \), a new class \( C \) is added, as well as the relationships \( C \leq_{cl} A \) and \( C \leq_{cl} B \). From Rule \( R1 \), it follows that \( M_2 = \{ (o \text{ type } B), (o \text{ type } C) \} \). In
$K_3$, class $\mathcal{B}$ is removed. Thus, $M_3 = \{(\text{o type C})\}$. In scenario (II), class $\mathcal{B}$ is removed and a new class $\mathcal{C}$ is added, along with the relationship $\mathcal{C} \leq_{cl} \mathcal{A}$. Then, since none of the rules $R1 - R5$ applies, it follows that $M_3 = \emptyset$ and $P_3 = \{(\text{o type C})\}$. Note that the non-backwards compatible transitions in scenarios (I) and (II) lead to different results. □
Chapter 11

Related Work

In this chapter, we discuss about related work to ontology evolution and versioning and to fuzzy and probabilistic Semantic Web.

11.1 Ontology Evolution

There are several works on ontology evolution and versioning, for a recent overview see [12]. Below we describe some of these works and we compare them with our work.

[On Backwards Compatible Ontology Evolution]

Klein et al. [17] propose a versioning mechanism for reducing the problems caused by ontology evolution. They argue that ontology versioning is necessary because changes to ontologies may cause incompatibilities, and drive to situations where the new (changed) ontology cannot be used in place of its previous version. They list a number of artifacts that may depend on an ontology, and thus can become incompatible after ontological evolution, and data that conforms to an ontology is one of them. When an ontology is changed, data may get a different interpretation or may use terms that do not exist any more. The authors introduce various forms of compatibility and one of them is backwards compatibility. In the same direction, Xuan et al. [38] propose a model to deal with the problem of asynchronous ontology versions in the context of a materialized integration system, which is based on the principle of ontological continuity, which refers to the permanence of classes, properties, and subsumption. This principle is actually what we
call backwards compatibility.

[Ontology Evolution and Data Validity]

Stojanovic et al. [31] identify a six-phase ontology evolution process and focus on providing the user with capabilities to control and customize it. In order to enable such customization of the ontology evolution process, the user may choose an advanced evolution strategy. It represents a mechanism to prioritize and arbitrate among different evolution strategies available in a particular situation, relieving the user of choosing elementary evolution strategies individually. Advanced evolution strategy automatically combines available elementary evolution strategies to satisfy user’s criteria. One of the adopted advanced evolution strategies is the *Instance-driven Evolution Strategy*. Noy and Klein [24] present an informal discussion on the differences between ontology evolution and database schema evolution, and how structural changes in ontologies affect the preservation of their data instances. They focus on whether instance data can still be accessed through the changed ontology, and they classify the operation effects as information-preserving changes, translatable changes, and information-loss changes. Now Konstaninidis et al. [19] focus on the *effects* of a requested change operation, i.e. how the new ontology version should be after a request for a change, and on its *side-effects* on the instance data, i.e. certain additional actions executed to restore validity. They propose a general-purpose algorithm for determining the effects and side-effects of a requested elementary or complex change operation, and such works can be used to resolve the conflicts. In addition, Qin and Alturi [26] focus on the validity issue of data instances during ontological evolution. They classify the changes to ontologies into two levels - structural and semantic. Semantic changes are brought by structural changes and can be further classified into explicit and implicit changes. They propose an algorithm for evaluating the structural validity of a data instance and then another algorithm for evaluating the semantic validity of a data instance.

Note that, in our work, we do not focus on the instance validity caused by ontology evolution. We consider that the ontology evolution does not effect the instance triples. In the case where a class \( c \) is deleted from one ontology \( S_K \), we consider that, in the new ontology \( S_{K'} \), the instances \( o \) that were explicitly classified in \( c \), i.e. \( (o \text{ type } c) \in I_K \), are
not affected. Thus, $c$ remains in $K'$ as an unconnected class. Now, in the case where a property $pr$ is deleted from one ontology $S_K$ (and also the statements regarding its domain and range are deleted from the initial schema), we consider that, in the new ontology $S_{K'}$, the property instance triples $(o \ pr \ o') \in I_K$ are not affected. Thus, $pr$ remains in $K'$ as an unconnected property and its domain and range is the top class, i.e. Resource. In future work, it would be worth to investigate, in the non-backwards compatible schema evolution case, a possible combination of the “repair” of invalid instance descriptions proposed in [19] with our proposal for computing possible instance descriptions. In conclusion, there are several works and approaches for dealing with the validity of data during migration, however there is no work for managing their specificity and quality while ontologies evolve.

11.2 Probabilistic and Uncertain Information

There are several works on probabilistic and uncertain information. Below we discuss about some of these works and make comments on how we could combine them with our work.

[Fuzzy/Probabilistic Semantic Web]

At last we could also say that our work is complementary to the works that have been proposed recently regarding fuzzy or probabilistic Semantic Web. Below, we describe and comment on some of these approaches. First, we describe extensions with uncertain information of the web ontology language RDF. For instance, Udrea et al. [36] introduce a Probabilistic RDF framework (for short pRDF) for expressing probabilistic information about the relationships expressed in RDF, and provide algorithms to efficiently answer queries over pRDF ontologies. Mazzieri and Dragoni [22] present an extended syntax to represent fuzzy membership values within RDF statements, and elaborate on their interpretation. Straccia [32] describes a system for a fragment of fuzzy RDF, and shows how top-k fuzzy disjunctive queries can be answered by relying on the closure computation and top-k database engines. Huang and Liu [14] present a general framework for supporting SPARQL queries on probabilistic RDF databases, and a query evaluation framework based on possible world semantics. In general, we could say that the “output” of our work, i.e. the possibilities (or probabilities, if quantified appropriately), can be considered as
“input” to such frameworks, which one could use for probabilistic query answering.

[Uncertainty and OWL (Web Ontology Language)]

Now, we describe extensions with uncertain information of the web ontology language OWL. Ding et al. [7] propose a Bayesian network-based extension to OWL. To indicate the probabilistic extension, the authors introduce three kinds of OWL classes, and they define a set of translation rules for converting the probabilistically extended OWL ontology into the directed acyclic graph (DAG) of a Bayesian Network (BN). Each node in the DAG represents a variable and is associated with a conditional probability table (CPT), which defines the probability of each possible value of the node, given each combination of values for the node’s parents in the DAG. Again, we can say that the “output” of our work can be used as an input to the above work since RDF is a sublanguage of OWL. Costa et al. [5] extend OWL with uncertainty based on first-order Bayesian logic. The possibilities defined by our approach can be expressed as a set of knowledge structures (called MFrags) which represent probabilistic knowledge about a collection of related hypotheses. Scharrenbach and Bernstein [29] introduce Fuzzy OWL, Markov Logic, and Probabilistic Description Logics (PDL) for handling uncertain data and resolving inconsistencies. They propose the concept of defaults which are specific constraints in PDL that can be used in order to remove incoherence in OWL ontologies. PDL distinguishes between terminological and assertional probabilistic knowledge, so the authors present a KB which represents the (assertional) probabilistic knowledge as our eKB does. Koller et al. [18] and Giungo et al. [9] propose probabilistic extensions of Description Logics (DLs). Koller et al. [18] present P-CLASSIC, which is a probabilistic extension of the description logic CLASSIC. Giungo et al. [9] develop a probabilistic extension of DAML+OIL for representing and reasoning with probabilistic ontologies in the Semantic Web (SW). They define P-SHOQ(D) as the probabilistic extension of SHOQ-(D), which is the description logic that provides a formal semantics, and a reasoning support for DAML+OIL. In [21], Lukasiewicz presents the expressive probabilistic logics P-SHIF(D) and P-SHOIN(D), which are probabilistic extensions of the corresponding description logics. These logics allow for expressing rich terminological probabilistic knowledge about concepts and roles, as well as assertional probabilistic knowledge about instances of concepts and roles. The
author presents sound and complete algorithms for the main reasoning problems in the new probabilistic description logics, which are based on reductions to reasoning in their classical counterparts, and to solving linear optimization problems. These works concern Description Logics (DLs). As our work concerns RDF, most of what is proposed in these works goes beyond the scope of this thesis.

Synopsizing, the above works introduce language constructs for representing uncertain information, and define the semantics and reasoning services of such knowledge bases. Our work is complementary, in the sense that we propose a method for deriving uncertainties based on the evolution of ontologies.

Finally, we should mention the W3C Uncertainty Reasoning for the World Wide Web (URW3) Incubator group\(^1\) whose mission is to better define the challenge of reasoning with, and representing uncertain information available through the World Wide Web and related WWW technologies.

\(^1\)http://www.w3.org/2005/Incubator/urw3/
Chapter 12

Concluding Remarks

Current approaches and techniques for ontology evolution, ignore that ontology evolution apart from conflicts it can decrease the specificity of the descriptions that have been defined using past ontology versions. The rapid evolution of ontologies requires principles, techniques, and tools for managing the quality of the migrated descriptions, as well as flexible interactive methods for managing this kind of uncertainty. To the best of our knowledge this is the first work that exploits ontology evolution for managing the specificity of instance descriptions. Specifically, in this work we formalized the problem with the notion of $X$-partition of the set of cartesian instance triples ($B_K$) of a KB $K$, and we defined the principles and rules that specify how $X$-partitions should be updated after instance migrations in ontology evolution. We provided two algorithms that compute the new set of possible instance triples $P_K'$ based just on the previous version of $K$, the previous $P_K$, and the new set of schema triples $S_K'$. Specifically, the first algorithm (Algorithm 2) concerns backwards compatible schema evolution, while the second algorithm (Algorithm 7) concerns non-backwards compatible schema evolution. Algorithm 7 is more general than Algorithm 2 and it applies even in the backwards compatible schema evolution case. However, Algorithm 2 has less steps and is more efficient. Since the ultimate objective is not just the identification of possibilities, but to aid making the instance descriptions as specific as possible, we proposed a specificity lifecycle management process that prompts to the user a subset of the possible instance triples (according to certain criteria) and showed how the extended KB ($eKB$) should be updated when the user approves or rejects some of them. Subsequently, we showed how possible instance triples can
be quantified and ranked for aiding users during the specificity life-cycle process. Further, we presented a compact (interval-based) representation of the possible instance triples $P_K$, appropriate for very large data sets, and a prototype system, called RIMQA. Finally, we applied our approach on real and synthetic datasets for demonstrating the feasibility of our approach.

In the future, we plan to generalize our approach of possibilities to the XSD-typed literal values of property instance triples [25]. Additionally, we plan to investigate a possible combination of the “repair” of invalid instance descriptions proposed in [19] with our proposal for computing possible instance descriptions. Further, we plan to improve our implementation by supporting a graphical visualization of the suggested possibilities. Finally, we plan to extend our theory such that disjointness conditions between classes are supported.

\footnote{The compact representation of $P_K$ is based on Lemmas 2(1) and 2(2).}
Bibliography


Appendix A: Proofs

In this Appendix, we provide the proof of the Lemmas and Propositions, presented in this thesis.

Lemma 1

1. If \((o \text{ type } c_2) \in P_K\) and \(c_2 \leq^*_cl c_1\) then \((o \text{ type } c_1) \in (P_K \cup C_i(K))\).

2. If \((o \text{ pr}_2 o') \in P_K\) and \(pr_2 \leq^*_pr pr_1\) then \((o \text{ pr}_1 o') \in (P_K \cup C_i(K))\).

Proof:

1) Let \((o \text{ type } c_2) \in P_K\) and \(c_2 \leq^*_cl c_1\). We will show that \((o \text{ type } c_1) \in (C_i(K) \cup P_K)\).

Certainly, \(o \in \text{Inst}_K \cap \text{URI}\). Assume that \((o \text{ type } c_1) \notin (C_i(K) \cup P_K)\). Since \(o \in \text{Inst}_K \cap \text{URI}\) and \(c_1 \in C_K\), it follows that \((o \text{ type } c_1) \in B_K\). Therefore, it follows from the definition of X-partition (Def. 5) that \((o \text{ type } c_1) \in M_K\). It follows from (ii) of Def. 5 that \((o \text{ type } c_2) \in M_K\). Thus, \((o \text{ type } c_2) \notin P_K\), which is impossible. Therefore, \((o \text{ type } c_1) \in (C_i(K) \cup P_K)\).

2) Let \((o \text{ pr}_2 o') \in P_K\) and \(pr_2 \leq^*_pr pr_1\). We will show that \((o \text{ pr}_1 o') \in (C_i(K) \cup P_K)\).

Since \((o \text{ pr}_2 o') \in P_K\), it holds that \(o \in \text{Inst}_K \cap \text{URI}\) and \(o' \in \text{Inst}_K\). Therefore, \((o \text{ pr}_1 o') \in B_K\). Assume that \((o \text{ pr}_1 o') \notin (C_i(K) \cup P_K)\). Therefore, it follows from the definition of X-partition (Def. 5) that \((o \text{ pr}_1 o') \in M_K\). It follows from (ii) of Def. 5 that \((o \text{ pr}_2 o') \in M_K\). Thus, \((o \text{ pr}_2 o') \notin P_K\), which is impossible. Therefore, \((o \text{ pr}_1 o') \in (C_i(K) \cup P_K)\). \(\square\)

Lemma 2

1. If \(c_1 \leq^*_cl c_2 \leq^*_cl c_3\) and \((o \text{ type } c_1), (o \text{ type } c_3) \in P_K\) then \((o \text{ type } c_2) \in P_K\).

2. If \(pr_1 \leq^*_pr pr_2 \leq^*_pr pr_3\) and \((o \text{ pr}_1 o'), (o \text{ pr}_3 o') \in P_K\) then \((o \text{ pr}_2 o') \in P_K\).
Proof:

1) Let \( c_1 \leq_{cl}^* c_2 \leq_{cl}^* c_3 \) and \( (o \text{ type } c_1), (o \text{ type } c_3) \in P_K \), we will show that \( (o \text{ type } c_2) \in P_K \). Note that \( o \in \text{Inst}_K \cap \text{URI} \) and \( c_2 \in C_K \). Thus, \( (o \text{ type } c_2) \in B_K \). Assume that \( (o \text{ type } c_2) \notin P_K \) then \( (o \text{ type } c_2) \in M_K \) or \( (o \text{ type } c_2) \in C_i(K) \). If \( (o \text{ type } c_2) \in M_K \) then it follows from (ii) of Def. 5 that \( (o \text{ type } c_1) \in M_K \), which is impossible. If \( (o \text{ type } c_2) \in C_i(K) \) then it follows from the RDF/S semantics that \( (o \text{ type } c_3) \in C_i(K) \), which is also impossible. Thus, \( (o \text{ type } c_2) \in P_K \).

2) Let \( pr_1 \leq_{pr}^* pr_2 \leq_{pr}^* pr_3 \) and \( (o \text{ pr } 1 \ o'), (o \text{ pr } 3 \ o') \in P_K \) we will show that \( (o \text{ pr } 2 \ o') \in P_K \). Note that \( o \in \text{Inst}_K \cap \text{URI} \), \( o' \in \text{Inst}_K \), and \( pr_2 \in Pr_K \). Thus, \( (o \text{ pr } 2 \ o') \in B_K \). Assume that \( (o \text{ pr } 2 \ o') \notin P_K \). Then, \( (o \text{ pr } 2 \ o') \in M_K \) or \( (o \text{ pr } 2 \ o') \in C_i(K) \). If \( (o \text{ pr } 2 \ o') \in M_K \) then it follows from (ii) of Def. 5 that \( (o \text{ pr } 1 \ o') \in M_K \), which is impossible. If \( (o \text{ pr } 2 \ o') \in C_i(K) \) then it follows from the RDF/S semantics that \( (o \text{ pr } 3 \ o') \in C_i(K) \). Therefore, \( (o \text{ pr } 2 \ o') \in P_K \). \( \square \)

Prop. 1 When a KB \( K = (S_K, I_K) \) evolves to a new KB \( K' = (S_{K'}, I_{K'}) \), where \( I_K = I_{K'} \), it holds that \( \text{Inst}_K = \text{Inst}_{K'} \).

Proof: It holds that \( \text{Inst}_K = \text{Res}_K \setminus (C_K \cup Pr_K) \). Since \( K = (S_K, I_K) \) and from the way that instance triples and schema triples are defined, it follows that \( \text{Inst}_K = \{ o \mid (o \text{ type } c) \in I_K \} \cup \{ o, o' \mid (o \text{ pr } o') \in I_K \} \). Since \( I_K = I_{K'} \). It follows that \( \text{Inst}_K = \text{Inst}_{K'} \). \( \square \)

Prop. 2 If \( S_K \subseteq S_{K'} \) then \( B_K \subseteq B_{K'} \).

Proof: Let \( b \in B_K \). If \( b \) is a class instance triple of the form \( (o \text{ type } c) \) then \( o \in \text{Inst}_K \cap \text{URI} \) and \( c \in C_K \). Since \( \text{Inst}_K = \text{Inst}_{K'} \), it follows that \( o \in \text{Inst}_{K'} \cap \text{URI} \). Additionally, since \( S_K \subseteq S_{K'} \), it holds that \( c \in C_{K'} \). Thus, \( b \in B_{K'} \). If \( b \) is a property instance triple of the form \( (o \text{ pr } o') \) then \( o \in \text{Inst}_K \cap \text{URI} \), \( o' \in \text{Inst}_K \), and \( pr \in Pr_K \). Thus, \( o \in \text{Inst}_{K'} \cap \text{URI} \), \( o' \in \text{Inst}_{K'} \) and, since \( S_K \subseteq S_{K'} \), it holds that \( pr \in Pr_{K'} \). Thus, \( b \in B_{K'} \). \( \square \)
Prop. 3 Let $S$ and $S'$ be two sets of schema triples. It holds that: $S \subseteq S'$ iff $\Delta_d(S \rightarrow S')$ contains only add operations.

Proof:
\[ \Rightarrow \] $\Delta_d(S \rightarrow S') = \{\text{Add}(t) \mid t \in S' \setminus C(S)\} \cup \{\text{Del}(t) \mid t \in S \setminus C(S')\}$. We have to prove that $S \setminus C(S') = \emptyset$. It is known from the closure definition that $S \subseteq C(S)$. $S \subseteq S'$ means that $C(S) \subseteq C(S')$, so by transitivity we get that $S \subseteq C(S')$.

\[ \Leftarrow \] $S \setminus C(S') = \emptyset$, so $S \subseteq C(S')$. Thus, $C(S) \subseteq C(C(S'))$. Therefore, $C(S) \subseteq C(S')$. By Definition 6, we get that $S \subseteq S'$.

Prop. 4 In the context of a transition $(C_i(K), M_K, P_K) \leadsto (C_i(K'), M_{K'}, P_{K'})$, it follows that: $M_K \cap P_{K'} = \emptyset$ (II2) iff $(M_K \setminus C_i(K')) \subseteq M_{K'}$.

Proof:
\[ \Rightarrow \] Assume that $M_K \cap P_{K'} = \emptyset$. Let $t \in M_K \setminus C_i(K')$ and assume that $t \notin M_{K'}$. Since $S_K \subseteq S_{K'}$, it holds that $t \in B_{K'}$. Therefore, based on the definition of X-partition (Def. 5) and the fact $t \notin C_i(K')$, it holds that $t \in P_{K'}$. However, in this case, it holds that $M_K \cap P_{K'} \neq \emptyset$, which is impossible. Therefore, $t \in M_{K'}$. Thus, $(M_K \setminus C_i(K')) \subseteq M_{K'}$.

\[ \Leftarrow \] Assume that $(M_K \setminus C_i(K')) \subseteq M_{K'}$. Further, assume that it exists $t \in M_K \cap P_{K'}$. Then, based on the definition of X-partition (Def. 5), it holds that $t \notin C_i(K')$. Thus, $t \in M_K \setminus C_i(K')$. Since $(M_K \setminus C_i(K')) \subseteq M_{K'}$, it follows that $t \in M_{K'}$. However, this is impossible, since $t \in P_{K'}$. Thus, $M_K \cap P_{K'} = \emptyset$.

Prop. 5 Consider an X-partition $(C_i(K), M_K, P_K)$ based on a schema $S_K$ and suppose we want to define the X-partition after migrating $I_K$ to a backwards compatible schema $S_{K'}$. We can derive $M_{K'}$ using the following rules:

(R1) If $(\text{o type } c) \in M_K$, $c' \leq^*_d c$, and $(\text{o type } c) \notin C_i(K')$ then $(\text{o type } c') \in M_{K'}$. 

115
(R2) If \((o \text{ pr } o') \in M_K, pr' \leq_{pr}^* pr\), and \((o \text{ pr } o') \notin C_i(K')\), then \((o \text{ pr'} o') \in M_{K'}\).

(R3) If \((o \text{ pr } o') \in B_{K'}\) and \(\neg \text{valid}(o, pr, o', K')\) then \((o \text{ pr'} o') \in M_{K'}\).

Proof: Rule R1 and Rule R2 follow directly from Prop. 4 and (ii) of Def. 5. Rule R3 follows directly from (iii) of Def. 5.

Prop. 6 The derivation of \(M_{K'}\) by the rules of Prop. 5 and of \(P_{K'}\) by Def. 8, yields a three-fold partition that is an X-partition (according to Def. 5) and respects postulates \(\Pi_1\) and \(\Pi_2\) of Def. 7.

Proof: The sets \(C_i(K')\), \(M_{K'}\), and \(P_{K'}\) are pairwise disjoint by construction. \(M_{K'}\) is lower set by construction, and contains all invalid property instance triples of \(B_{K'}\). Therefore, \((C_i(K'), M_{K'}, P_{K'})\) is an X-partition.

Postulate \(\Pi_1\) is satisfied by construction. What is left to prove is that \(\Pi_2\) is satisfied (past negative information cannot become possible), i.e. that \(M_K \cap P_{K'} = \emptyset\). Suppose this is not true, i.e. suppose there exist a \(t\) such that \(t \in M_K \cap P_{K'}\). Then, \(t \in M_K\) and \(t \notin C_i(K')\). Thus, due to Rule R1 and Rule R2 of Prop. 5, \(t\) (and its subtriples) would belong to \(M_{K'}\). Thus, \(t\) cannot belong to \(P_{K'}\) (since \(P_{K'}\) as defined by Def. 8 excludes all elements of \(M_{K'}\)).

Prop. 7 For a new class \(c' \in C_{K'} \setminus C_K\), it holds that: \((o \text{ type } c') \in P_{K'}\) iff

(i) \(o \in \text{Inst}_K \cap \text{URI}\),

(ii) for all \(c \in C_K\) s.t. \(c' \leq_{\delta}^* c\), it holds that \((o \text{ type } c) \in (C_i(K') \cup P_K)\), and

(iii) \((o \text{ type } c') \notin C_i(K')\).

Proof:

\(\Rightarrow\) Let \((o \text{ type } c') \in P_{K'}\). Certainly, \(o \in \text{Inst}_K \cap \text{URI}\). We will show that for all \(c \in C_K\) s.t. \(c' \leq_{\delta}^* c\), it holds that \((o \text{ type } c) \in (C_i(K') \cup P_K)\) and \((o \text{ type } c') \notin C_i(K')\). Assume that it exists \(c \in C_K\) s.t. \((c' \leq_{\delta}^* c\) and \((o \text{ type } c) \notin (C_i(K') \cup P_K)\)) or \((o \text{ type } c') \in C_i(K')\). If it exists \(c \in C_K\) s.t. \((c' \leq_{\delta}^* c\) and \((o \text{ type } c) \notin (C_i(K') \cup P_K)\) then since \(C_i(K) \subseteq C_i(K')\), it follows that \((o \text{ type } c) \notin (C_i(K) \cup P_K)\). Since \(o \in \text{Inst}_K \cap \text{URI}\) and \(c \in C_K\), it follows that
(o type c) ∈ B_K. Therefore, it follows from Def. 5 that (o type c) ∈ M_K. It follows from Rule R1 of Prop. 5 that (o type c') ∈ M_{K'}. Thus, (o type c') ∉ P_{K'}, which is impossible. If (o type c') ∈ C_i(K'), it follows from Def. 5 that (o type c') ∉ P_{K'}, which is impossible.

⇐ Assume that o ∈ Inst_K ∩ URI and that for all c ∈ C_K s.t. c' ≤_c c, it holds that (o type c) ∈ (C_i(K') ∪ P_K). Additionally, assume that it holds that (o type c) ∈ (C_i(K') ∪ P_K). We will show that (o type c') ∈ P_{K'}. It follows from Def. 5 that for all c ∈ C_K s.t. c' ≤_c c, it holds that (o type c') ∈ B_K \ M_K or (o type c) ∈ C_i(K'). It follows that (o type c') ∉ M_{K'} (note Rule R1 of Prop. 5 does not apply). Since o ∈ Inst_K ∩ URI, c' ∈ C_{K'}, and Inst_K = Inst_{K'}, it follows that (o type c') ∈ B_{K'}. Now, since (o type c') ∉ C_i(K'), it follows from Def. 5 that (o type c') ∈ P_{K'}.

Prop. 8 For a new property pr' ∈ Pr_{K'}, it holds that: (pr' o') ∈ P_{K'} iff:

(i) o ∈ URI and valid(o, pr', o', K'),
(ii) for all pr ∈ Pr_K s.t. pr' ≤_pr pr, it holds that (o pr o') ∈ (C_i(K') ∪ P_K), and
(iii) (o pr' o') ∉ C_i(K').

Proof:

⇒) Let (o pr' o') ∈ P_{K'}. Since (o pr' o') ∈ P_{K'}, it follows that valid(o, pr', o', K').

Additionally, we have o ∈ URI. We will show that for all pr ∈ Pr_K s.t. pr' ≤_pr pr, it holds that (i) (o pr o') ∈ (C_i(K') ∪ P_K) and (ii) (o pr' o') ∉ C_i(K'). Assume that it exists pr ∈ Pr_K s.t. (i) pr' ≤_pr pr and (o pr o') ∉ (C_i(K') ∪ P_K), or (ii) (o pr' o') ∈ C_i(K'). If it exists pr ∈ Pr_K s.t. pr' ≤_pr pr and (o pr o') ∉ (C_i(K') ∪ P_K) then, since C_i(K) ⊆ C_i(K'), it follows that pr' ≤_pr pr' and (o pr o') ∉ (C_i(K)∪P_K). Obviously, it holds that (o pr o') ∈ B_K. Then, it follows from Def. 5 that (o pr o') ∈ M_K. Therefore, it follows from Rule R2 of Prop. 5 that (o pr o') ∈ M_{K'}. Thus, (o pr o') ∉ P_{K'}, which is impossible. If (o pr' o') ∈ C_i(K'), it follows from Def. 5 that (o pr o') ∉ P_{K'}, which is impossible.

⇐) Assume that (i) o ∈ URI and valid(o, pr', o', K'), (ii) for all pr ∈ Pr_K s.t. pr' ≤_pr pr, it holds that (o pr o') ∈ (C_i(K') ∪ P_K), and (iii) (o pr' o') ∉ C_i(K'). We will show that (o pr' o') ∈ P_{K'}. It follows from Def. 5 that for all pr ∈ Pr_K s.t. pr' ≤_pr pr, it holds that (o pr o') ∈ B_K \ M_K or (o pr' o') ∈ C_i(K'). It follows that (o pr' o') ∉ M_{K'} (note that Rule R2 and Rule R3 of Prop. 5 do not apply). It holds that o ∈ Inst_{K'} ∩ URI, o' ∈ Inst_{K'},

117
$pr \in Pr_{K'}$. Thus, $(o\ pr'\ o') \in B_{K'}$. Since $(o\ pr'\ o') \notin C_i(K')$, it follows from Def. 5 that $(o\ pr'\ o') \in P_{K'}$. 

\textbf{Prop. 9} Let $K = (S_K, I_K)$ be a KB and let $S_{K'}$ be a set of new schema triples s.t. $S_K \subseteq S_{K'}$ and $K' = (S_{K'}, I_K)$ be a (valid) KB then $P_{K'} = Produce\_Possibilities(K, P_K, S_{K'})$.

\textbf{Proof:} Initially, $P_{K,Add} = \emptyset$ and $P_{K,Del} = \emptyset$. For each new class $c' \in NC$, Algorithm $Produce\_Possibilities(K, P_K, S_{K'})$ inserts to $P_{K,Add}$ exactly the class instance triples indicated by Prop. 7. For each old class $c_1 \in C_K$, if it exists $c_2 \in C_K$ s.t. $c_1 \leq_{cl}^* c_2$ and $(o\ type\ c_2) \notin (P_K \cup C_i(K'))$ then all class instance triples $(o\ type\ c_1) \in P_K$ are added to $P_{K,Del}$. Since $S_K \subseteq S_{K'}$, it holds that $(o\ type\ c_2) \notin C_i(K)$. Thus, $(o\ type\ c_2) \in B_K \setminus (P_K \cup C_i(K)) = M_K$ and $(o\ type\ c_2) \notin C_i(K')$. Therefore, due to Rule R1 of Prop. 5, it holds that $(o\ type\ c_1) \in M_{K'}$.

Similarly, for each new property $pr' \in NP$, Algorithm $Produce\_Possibilities(K, P_K, S_{K'})$, inserts to $P_{K,Add}$ exactly the property instance triples indicated by Prop. 8. For each old property $pr_1 \in Pr_K$, if it exists $pr_2 \in Pr_K$ s.t. $pr_1 \leq_{pr}^* pr_2$ and $(o\ pr_2\ o') \notin (P_K \cup C_i(K'))$ then all class instance triples $(o\ pr_1\ o') \in P_K$ are added to $P_{K,Del}$. Since $(o\ pr_1\ o') \in P_K$, it holds that $(o\ pr_1\ o') \in B_K$. Additionally, since $S_K \subseteq S_{K'}$, it holds that $(o\ pr_2\ o') \notin C_i(K)$. Therefore, $(o\ pr_2\ o') \in B_K \setminus (P_K \cup C_i(K)) = M_K$ and $(o\ pr_2\ o') \notin C_i(K')$. Thus, due to Rule R2 of Prop. 5, it holds that $(o\ pr_1\ o') \in M_{K'}$.

All instance triples in $P_K \cap C_i(K')$ are moved to $P_{K,Del}$. Finally, $P_{K'} = (P_K \setminus P_{K,Del}) \cup P_{K,Add}$. 

\textbf{Prop. 10} The time complexity of Algorithm 2 is $O(|Inst_K|^2 |K'|^2 * (|K'|^2 + |P_K|))$.

\textbf{Proof:} First, we will prove the following Lemma.

\textbf{Lemma:} Let $K$ be a KB. Then, (i) the size complexity of $C(K)$ is in $O(|K|^2)$ and (ii) the time complexity of $C(K)$ is in $O(|K|^4)$.

\textbf{Proof:}

(i) First note that $K \subseteq C(K)$. Let $T_{rdfs}$ denote the RDF and RDFS axiomatic triples [13],
The number of \( \text{pr} \) terms, for \( i \in \{1, 2, \ldots \} \). It holds that \( T_{rdfs} \subseteq C(K) \).

The inference rules used for the derivation of \( C(K) \) are the following:

1. If \( (c \leq_{cl} c') \in K \cup T_{rdfs} \) then \( (c \leq_{cl} c') \in C(K) \).
2. If \( (c \text{ type Class}) \in C(K) \) then \( (c \leq_{cl} c) \in C(K) \).
3. If \( (c_1 \leq_{cl} c_2) \in C(K) \) and \( (c_2 \leq_{cl} c_3) \in C(K) \) then \( (c_1 \leq_{cl} c_3) \in C(K) \).
4. If \( (c \text{ type Class}) \in C(K) \) then \( (c \leq_{cl} \text{ Resource}) \in C(K) \).
5. If \( (o \text{ type } c) \in K \cup T_{rdfs} \) then \( (o \text{ type } c) \in C(K) \).
6. If \( (o \text{ type } c) \in C(K) \) and \( (c \leq_{cl} c') \in C(K) \) then \( (o \text{ type } c') \in C(K) \).
7. If \( (o \text{ pr } o') \in C(K) \) and \( (pr \text{ domain } c) \in C(K) \) then \( (o \text{ type } c) \in C(K) \).
8. If \( (o \text{ pr } o') \in C(K) \) and \( (pr \text{ range } c) \in C(K) \) then \( (o' \text{ type } c) \in C(K) \).
9. If \( (pr \leq_{pr} pr') \in K \cup T_{rdfs} \) then \( (pr \leq_{pr} pr') \in C(K) \).
10. If \( (pr \text{ type Property}) \in C(K) \) then \( (pr \leq_{pr} pr) \in C(K) \).
11. If \( (pr_1 \leq_{pr} pr_2) \in C(K) \) and \( (pr_2 \leq_{pr} pr_3) \in C(K) \) then \( (pr_1 \leq_{pr} pr_3) \in C(K) \).
12. If \( (o \text{ pr } o') \in K \cup T_{rdfs} \) and \( (pr \leq_{pr} pr') \in C(K) \) then \( (o \text{ pr } o') \in C(K) \).
13. If \( (o \text{ pr } o') \in K \cup T_{rdfs} \) then \( (o \text{ type } \text{ Resource}), (pr \text{ type } \text{ Resource}), (o' \text{ type } \text{ Resource}) \in C(K) \).
14. If \( (o \text{ pr } o') \in K \cup T_{rdfs} \) then \( (pr \text{ type } \text{ Property}) \in C(K) \).
15. If \( (c \leq_{cl} c') \in K \cup T_{rdfs} \) then \( (c' \text{ type Class}) \in C(K) \).
16. If \( (pr \leq_{cl} pr) \in K \cup T_{rdfs} \) then \( (pr \text{ type Property}) \), \( (pr' \text{ type Property}) \in C(K) \).
17. If \( (pr \text{ rdfs:domain } c) \in K \cup T_{rdfs} \) then \( (pr \text{ type Property}) \), \( (c \text{ type rdfs:Class}) \in C(K) \).
18. If \( (pr \text{ range } c) \in K \cup T_{rdfs} \) then \( (pr \text{ type Property}) \), \( (c \text{ type rdfs:Class}) \in C(K) \).

The number of \( (c \leq_{cl} c') \) triples in \( C(K) \) due to rules (1-4) is in \( O(|C_K|^2) = O(|K|^2) \).

The number of \( (o \text{ type } c') \) triples in \( C(K) \) due to rules (5-8) is in \( O(|K| \cdot |C_K|) = O(|K|^2) \).

The number of \( (pr \leq_{pr} pr') \) triples in \( C(K) \) due to rules (9-11) is in \( O(|Pr_K|^2) = O(|K|^2) \).

The size complexity due to rules (12) is in \( O(|K| \cdot |Pr_K|) = O(|K|^2) \).

The total size complexity of \( C(K) \) is in \( O(|K|^2) \).

(ii) Based on the form of the inference rules i(1)-i(18) and the form of the KBs that we consider, it follows that we can compute \( C(K) \) if (i) we execute the rules i(1) and i(3) until
fixpoint, (ii) we execute the rules i(9) and i(11) until fixpoint, (iii) we execute the rules i(13)-i(18), (iv) we execute the rules i(5) and i(12), (v) we execute the rules i(7)-i(8), (vi) we execute the rule i(4), (vii) we execute the rule i(6), and finally (viii) we execute the rules i(2) and i(10). The time complexity of the steps (i) and (ii) is \(O(|K|^3)\), because the time complexity to compute the transitive closure of \(\leq_{cl}\) and \(\leq_{pr}\) is \(O(|K|^3)\) [15]. Based on the fact that the size complexity of \(C(K)\) is in \(O(|K|^2)\), the time complexity of the rest of the steps is \(O(|K|^4)\).

**End Of Lemma**

Since \(S_K \subseteq S_{K'}\), it holds that \(|K| \leq |K'|\), assuming that the representation of \(S_K\) and \(S_{K'}\) is redundancy-free, i.e. only the reflexive and transitive reduction of the subClassOf and subPropertyOf relationships are stated in \(K\) and \(K'\). Now, we will compute the time complexity of lines (5-6) of the Algorithm. The time for the tests \((o\ type\ c) \in (C_i(K') \cup P_K)\) in line (6) is \(O(|NC| * |Inst_K| * |C_K| * (|K'|^2 + |P_K|))\). The time for the tests in line (6) \((o\ type\ c') \notin C_i(K')\) is \(O(|NC| * |Inst_K| * |K'|^2)\). The time for computing \(Inst_K \cap URI\) in line (6), is \(O(|Inst_K|)\). The time for computing \(\{c \mid c' \leq_{d}^* c\}\), for all \(c' \in NC\), is \(O(|NC| * |K'|^2)\) and the time of finding which of the elements of these sets belong to \(C_K\) is \(O(|NC| * |C_{K'}| * |K'|) = O(|NC| * |K'| * |K|)\). Thus, the total time complexity for the lines (5-6) is: \(O(|NC| * |Inst_K| * |C_K| * (|K'|^2 + |P_K|)) = O(|Inst_K| * |K| * |K'| * (|K'|^2 + |P_K|))\).

Now, we will compute the time complexity of lines (7-8). The time for computing \(\{c_2 \mid c_1 \leq_{d} c_2\}\), for all \(c_1 \in C_K\), is \(O(|C_K| * |K'|^2)\) and the time of finding which of the elements of this sets belong to \(C_K\) is \(O(|C_K| * |C_{K'}| * |K|)\). The time for checking which of the instance triples \((o\ type\ c_2) \notin P_K \cup C_i(K')\) is \(|C_K| * |Inst_K| * |C_K| * (|K'|^2 + |P_K|)\). The time for checking which of the instance triples \((o\ type\ c_1) \in P_K\) is \(|C_K| * |Inst_K| * |P_K|\). Thus, the time complexity of lines (7-8) is \(O(|Inst_K| * |K|^2 * (|K'|^2 + |P_K|))\).

Now, we will compute the time complexity of lines (10-11). The time for the tests \((o\ pr\ o') \in (C_i(K') \cup P_K)\) in line (11) is \(O(|NP| * |Inst_K| * |Inst_K| * |Pr_K| * (|K'|^2 + |P_K|))\). The time for the tests in line (11) \((o\ pr'\ o') \notin C_i(K')\) is \(O(|NP| * |Inst_K| * |Inst_K| * |K'|^2)\). The time for computing \(\{pr \mid pr' \leq_{pr}^* pr\}\), for all \(pr' \in NP\), is \(O(|NP| * |K'|^2)\) and the time of finding which of the elements of these sets belong to \(Pr_K\) is \(O(|NP| * |Pr_K| * |K|) = O(|NP| * |K'| * |K|)\). Thus, the total complexity time for the lines (10-11) is: \(O(|NP| * |Inst_K|^2 * |Pr_K| * ((|K'|^2 + |P_K|)) = O(|Inst_K|^2 * |K| * |K'| * (|K'|^2 + |P_K|))\).
Now, we will compute the time complexity of lines (12-13). The time for computing \( \{pr_2 \mid pr_1 \leq_{pr} pr_2\} \), for all \( pr_1 \in Pr_K \), is \( O(|Pr_K| \cdot |K'|^2) \) and the time of finding which of the elements of these sets belong to \( Pr_K \) is \( O(|Pr_K| \cdot |Pr_{K'}| \cdot |K|) \). The time for checking which of the instance triples \((o \ pr_2 \ o') \notin P_K \cup C_i(K')\) is \( O(|Pr_K| \cdot |Inst_K|^2 \cdot |Pr_K| \cdot (|K'|^2 + |P_K|)) \). The time for checking which of the instance triples \((o \ pr_1 \ o') \in P_K\) is \( O(|Pr_K| \cdot |Inst_K|^2 \cdot |P_K|) \). Thus, the total time complexity of lines (12-13) is \( O(|Inst_K|^2 \cdot |K|^2 \cdot (|K'|^2 + |P_K|)) \).

The time complexity of line (14) is \( O(|P_K| \cdot |K'|^2) \). Now, we will compute the time complexity of line (15). The size of \( P_{K,Del} \) is \( O(|Inst_K| \cdot |C_K| + |Inst_K|^2 \cdot |Pr_K|) = O(|Inst_K|^2 \cdot |K|) \). Thus, the time complexity of line (15) is \( O(|P_K| \cdot |Inst_K|^2 \cdot |K|) \).

As shown in Lemma, the time complexity of computing \( C(K) \) and \( C(K') \) is \( O(|K|^4) \) and \( O(|K'|^4) \), respectively.

Therefore, the total time complexity of Algorithm 2 is \( O(|Inst_K|^2 \cdot |K'|^2 \cdot (|K'|^2 + |P_K|)) \).

\[ \square \]

**Prop. 11:** Let \( X \subseteq P_K \). If \( K^{up} = K \cup X \) then \( C(K^{up}) = C(K) \cup SupTriples(X) \).

**Proof:** If \((o \ type \ c) \in X\), for \( c \in C_K\), then the newly derived triples in \( C(K^{up})\), due to this newly added RDF triple to \( K^{up}\), are \( \{(o \ type \ c') \mid c \leq_{cl} c'\} \). If \((o \ pr \ o') \in X\), for \( pr \in Pr_K\), then the newly derived triples in \( C(K^{up})\), due to this newly added RDF triple to \( K^{up}\), are \( \{(o \ pr' \ o') \mid pr \leq^*_{pr} pr'\} \cup \{(o \ type \ domain(pr)) \mid pr \leq^*_{pr} pr'\} \cup \{(o' \ type \ range(pr')) \mid pr \leq^*_{pr} pr'\} \). We will show that \( \{(o \ type \ domain(pr')) \mid pr \leq^*_{pr} pr'\} \cup \{(o' \ type \ range(pr')) \mid pr \leq^*_{pr} pr'\} \subseteq C(K) \). Since \((o \ pr' \ o') \in P_K\), it holds from (iii) of Def. 5 that \((o \ type \ domain(pr)) \in C(K)\) and \((o' \ type \ range(pr')) \in C(K)\). If it exists \( pr'\) s.t. \( pr \leq^*_{pr} pr'\) then, since \( K\) is a valid KB, it holds that (i) \( domain(pr) \leq^*_{cl} domain(pr')\) and (ii) \( range(pr) \leq^*_{cl} range(pr')\). Therefore, \((o \ type \ domain(pr')) \in C(K)\) and \((o' \ type \ range(pr')) \in C(K)\). \( \square \)
Prop. 12: Let $K$ be a KB. It holds that:

$$P_{comp}^K = P_1 \cup P_2 \cup P_3,$$

where:

$$P_1 = \{ \{ h_1, t \} \mid t = (o \text{ pr } o') \notin \text{Valid}(K) \text{ where } o, o' \in \text{Inst}_K, \text{pr} \in \text{Pr}_K, \quad h_1 = (o \text{ type } c_1) \in P_K, \text{ where } \text{domain(pr)} = c_1, \text{ range(pr)} \in \text{Cl}(o') \}$$

$$P_2 = \{ \{ t, h_2 \} \mid t = (o \text{ pr } o') \notin \text{Valid}(K) \text{ where } o, o' \in \text{Inst}_K, \text{pr} \in \text{Pr}_K, \quad h_2 = (o' \text{ type } c_2) \in P_K, \text{ where } \text{range(pr)} = c_2, \text{ domain(pr)} \in \text{Cl}(o) \}$$

$$P_3 = \{ \{ h_1, t, h_2 \} \mid t = (o \text{ pr } o') \notin \text{Valid}(K) \text{ where } o, o' \in \text{Inst}_K, \text{pr} \in \text{Pr}_K, \quad h_1 = (o \text{ type } c_1) \in P_K \text{ where } \text{domain(pr)} = c_1, \quad h_2 = (o' \text{ type } c_2) \in P_K \text{ where } \text{range(pr)} = c_2 \}$$

**Proof:** First, we will prove the following lemma.

**Lemma:** It holds that $\{ s \cup \{ t \} \mid s \subseteq P_K, t \notin \text{Valid}(K), t \in \text{Valid}(K) \cup s \}$, and $t \notin \text{Valid}(K \cup s') \forall s' \subset s = P_1' \cup P_2' \cup P_3'$ where

$$P_{comp}^K = P_1' \cup P_2' \cup P_3,$$

where:

$$P_1' = \{ \{ h_1, t \} \mid t = (o \text{ pr } o') \notin \text{Valid}(K) \text{ where } o, o' \in \text{Inst}_K, \text{pr} \in \text{Pr}_K, \quad h_1 = (o \text{ type } c_1) \in P_K, \text{ where } c_1 \leq_{cl}^* \text{domain(pr)}, \text{ range(pr)} \in \text{Cl}(o') \}$$

$$P_2' = \{ \{ t, h_2 \} \mid t = (o \text{ pr } o') \notin \text{Valid}(K) \text{ where } o, o' \in \text{Inst}_K, \text{pr} \in \text{Pr}_K, \quad h_2 = (o' \text{ type } c_2) \in P_K, \text{ where } c_2 \leq_{cl}^* \text{range(pr)}, \text{ domain(pr)} \in \text{Cl}(o) \}$$

$$P_3' = \{ \{ h_1, t, h_2 \} \mid t = (o \text{ pr } o') \notin \text{Valid}(K) \text{ where } o, o' \in \text{Inst}_K, \text{pr} \in \text{Pr}_K, \quad h_1 = (o \text{ type } c_1) \in P_K \text{ where } c_1 \leq_{cl}^* \text{domain(pr)}, \quad h_2 = (o' \text{ type } c_2) \in P_K \text{ where } c_2 \leq_{cl}^* \text{range(pr)}, \quad \text{domain(pr)} \notin \text{Cl}(o), \text{ and range(pr)} \notin \text{Cl}(o') \}$$

**Proof:**

$\Leftarrow$ Let $x \in P_1' \cup P_2' \cup P_3'$. Then, $x = s \cup \{ t \}$, where $t = (o \text{ pr } o') \notin \text{Valid}(K)$, for $o, o' \in \text{Inst}_K$ and $\text{pr} \in \text{Pr}_K$, and (i) $s = \{ (o \text{ type } c_1) \}$, where $c_1 \leq_{cl}^* \text{domain(pr)}$, if $x \in P_1'$, (ii) $s = \{ (o' \text{ type } c_2) \}$, where $c_2 \leq_{cl}^* \text{range(pr)}$, if $x \in P_2'$, and (iii) $s = \{ (o \text{ type } c_1), (o' \text{ type } c_2) \}$, where $c_1 \leq_{cl}^* \text{domain(pr)}$ and $c_2 \leq_{cl}^* \text{range(pr)}$, if $x \in P_3'$.
In all cases \( s \subseteq P_K \). Since \( \text{valid}(o, pr, o', K \cup s) \), it follows that \( t \in \text{Valid}(K \cup s) \).

It is easy to see that for all \( s' \subset s \), it holds that \( t \not\in \text{Valid}(K \cup s') \), because it does not hold \( \text{valid}(o, pr, o', K \cup s') \). Thus, \( x \in \{ s \cup \{ t \} \mid s \subseteq P_K, t \not\in \text{Valid}(K), t \in \text{Valid}(K \cup s), \) and \( t \not\in \text{Valid}(K \cup s') \forall s' \subset s \).

\( \Rightarrow \) Let \( x \in \{ s \cup \{ t \} \mid s \subseteq P_K, t \not\in \text{Valid}(K), t \in \text{Valid}(K \cup s), \) and \( t \not\in \text{Valid}(K \cup s') \forall s' \subset s \}. \) We will show that \( x \in P'_1 \cup P'_2 \cup P'_3 \). Let \( x = s \cup \{ t \} \). Since \( s \subseteq P_K, t \not\in \text{Valid}(K) \) and \( t \in \text{Valid}(K \cup s) \), it follows that \( t = (o \ pr \ o') \), where \( o, o' \in \text{Inst}_K \) and \( pr \in \text{Pr}_K \).

Since \( t \not\in \text{Valid}(K \cup s') \forall s' \subset s \), it follows that (i) if \( \text{range}(pr) \in \text{Cl}(o') \) then \( s = \{ h_1 \} \), where \( h_1 = (o \ type \ c_1) \in P_K \) and \( c_1 \leq^*_d \text{domain}(pr) \), (ii) if \( \text{domain}(pr) \in \text{Cl}(o) \) then \( s = \{ h_2 \} \), where \( h_2 = (o' \ type \ c_2) \in P_K \) and \( c_2 \leq^*_d \text{range}(pr) \), and (ii) if \( \text{domain}(pr) \not\in \text{Cl}(o) \) and \( \text{range}(pr) \not\in \text{Cl}(o') \) then \( s = \{ h_1, h_2 \} \), where \( h_1 = (o \ type \ c_1) \in P_K, h_2 = (o' \ type \ c_2) \in P_K, c_1 \leq^*_d \text{domain}(pr) \), and \( c_2 \leq^*_d \text{range}(pr) \). Therefore, it follows that \( x \in P'_1 \cup P'_2 \cup P'_3 \).

End of Lemma

From the above Lemma and Lemma 1(1), Proposition 12 now follows immediately. \( \square \)

**Prop. 13** In the context of a transition \( (C_i(K), M_K, P_K) \sim (C_i(K'), M_{K'}, P_{K'}) \), it follows that: \( (C_i(K) \setminus C_i(K')) \cap B_{K'} \subseteq M_{K'} \) (II3) iff \( C_i(K) \cap P_{K'} = \emptyset \).

**Proof:**

\( \Rightarrow \) Assume that \( (C_i(K) \setminus C_i(K')) \cap B_{K'} \subseteq M_{K'} \). Further, assume that it exists an instance triple \( t \in C_i(K) \cap P_{K'} \). Since \( t \in P_{K'} \), it follows that \( t \in B_{K'} \). Further, from the Definition of X-partition (Def. 5), it follows that \( t \not\in C_i(K') \). Thus, \( t \in (C_i(K) \setminus C_i(K')) \cap B_{K'} \). Therefore, \( t \in M_{K'} \), which is impossible. Thus, \( C_i(K) \cap P_{K'} = \emptyset \).

\( \Leftarrow \) Assume that \( C_i(K) \cap P_{K'} = \emptyset \) and let \( t \in (C_i(K) \setminus C_i(K')) \cap B_{K'} \). We will show that \( t \in M_{K'} \). Assume that \( t \not\in M_{K'} \). Then, from the Definition of X-partition (Def. 5), it follows that \( t \in P_{K'} \). Then, \( t \in C_i(K) \cap P_{K'} \). However, this is impossible since \( C_i(K) \cap P_{K'} = \emptyset \). Thus, \( t \in M_{K'} \). \( \square \)

**Prop. 14** For class and property instance triples:

(R4) If \( (o \ type \ c) \in C_i(K), c' \leq^*_d c \), and \( (o \ type \ c) \not\in C_i(K') \) then \( (o \ type \ c') \in M_{K'} \).
(R5) If \((o \text{ pr } o') \in C_i(K), pr' \leq_{pr}^* pr\), and \((o \text{ pr } o') \notin C_i(K')\) then \((o \text{ pr } o') \in M_{K'}\).

**Proof:** Rule R4 and Rule R5 follow directly from Postulate II3 of Def. 13 and (ii) of Def. 5. \(\square\)

**Prop. 15** The same as Prop. 7 but now applies to KBs \(K = (S_K, I_K)\) and \(K' = (S_{K'}, I_K)\), where \(S_K\) and \(S_{K'}\) are not necessarily backwards compatible.

**Proof:**
\(\Rightarrow\) Let \((o \text{ type } c') \in P_{K'}\). Certainly, \(o \in Inst_K \cap URI\). We will show that for all \(c \in C_K\) s.t. \(c' \leq_{cl} c\), it holds that \((o \text{ type } c) \in (C_i(K') \cup P_K)\) and \((o \text{ type } c') \notin C_i(K')\). Assume that it exists \(c \in C_K\) s.t. \((c' \leq_{cl} c\) and \((o \text{ type } c) \notin (C_i(K') \cup P_K)\)) or \((o \text{ type } c') \in C_i(K')\). If it exists \(c \in C_K\) s.t. \((c' \leq_{cl} c\) and \((o \text{ type } c) \notin (C_i(K') \cup P_K)\)) then it follows that \((o \text{ type } c) \notin (C_i(K) \cup P_K)\) or \(\((o \text{ type } c) \in C_i(K)\) and \((o \text{ type } c) \notin C_i(K')\)\). Since \(o \in Inst_K \cap URI\) and \(c \in C_K\), it follows that \((o \text{ type } c) \in B_K\). If \((o \text{ type } c) \notin (C_i(K) \cup P_K)\) then it follows from Def. 5 that \((o \text{ type } c) \in M_{K'}\). Therefore, it follows that \((o \text{ type } c') \in M_{K'}\) (see Rule R1 of Prop. 5). If \((o \text{ type } c) \in C_i(K)\) and \((o \text{ type } c) \notin C_i(K')\) then from Def. 14 (Rule R4) it follows that \((o \text{ type } c') \in M_{K'}\). Thus, \((o \text{ type } c') \notin P_{K'}\), which is impossible.

\(\Leftarrow\) Assume that \(o \in Inst_K \cap URI\) and that for all \(c \in C_K\) s.t. \(c' \leq_{cl} c\), it holds that \((o \text{ type } c) \in (C_i(K') \cup P_K)\). Additionally, assume that it holds \((o \text{ type } c') \notin C_i(K')\). We will show that \((o \text{ type } c') \in P_{K'}\). It follows from Def. 5 that for all \(c \in C_K\) s.t. \(c' \leq_{cl} c\), it holds that \((o \text{ type } c) \in B_K \setminus M_K\) or \((o \text{ type } c) \in C_i(K')\). It follows \((o \text{ type } c') \notin M_{K'}\) (note that Rule R1 of Prop. 5 and Rule R4 of Prop. 14 do not apply). Since \(o \in Inst_K \cap URI\), \(c' \in C_{K'}\), and \(Inst_K = Inst_{K'}\), it follows that \((o \text{ type } c') \in B_{K'}\). Now, since \((o \text{ type } c') \notin C_i(K')\), it follows from Def. 5 that \((o \text{ type } c') \in P_{K'}\). \(\square\)

**Prop. 16** The same as Prop. 8 but now applies to KBs \(K = (S_K, I_K)\) and \(K' = (S_{K'}, I_K)\), where \(S_K\) and \(S_{K'}\) are not necessarily backwards compatible.

**Proof:**
\(\Rightarrow\) Let \((o \text{ pr'} o') \in P_{K'}\). Since \((o \text{ pr'} o') \in P_{K'}\), it follows that \(\text{valid}(o, \text{ pr'}, o', K')\). We 124
will show that for all \( pr \in Pr_K \) s.t. \( pr' \leq_{PR} pr \), it holds that (i) \( (o \; pr \; o') \in (C_i(K') \cup P_K) \) and (ii) \( (o \; pr' \; o') \notin C_i(K') \). Assume that it exists \( pr \in Pr_K \) s.t. (i) \( (pr' \leq_{PR} pr \) and \( (o \; pr \; o') \notin (C_i(K') \cup P_K) \)), or (ii) \( (o \; pr' \; o') \in C_i(K') \). It follows that \( pr' \leq_{PR} pr \), and \( (o \; pr' \; o') \notin (C_i(K) \cup P_K) \) or \( ((o \; pr \; o') \in C_i(K) \) and \( (o \; pr' \; o') \notin C_i(K') \)). Obviously, \( (o \; pr' \; o') \in B_K \). If \( (o \; pr' \; o') \notin (C_i(K) \cup P_K) \) then it follows from Def. 5 that \( (o \; pr' \; o') \in M_K \).

Therefore, since \( (o \; pr' \; o') \notin C_i(K') \), it follows that \( (o \; pr' \; o') \in M_{K'} \) (see Rule R2 of Prop. 5). If \( (o \; pr' \; o') \in C_i(K) \) and \( (o \; pr' \; o') \notin C_i(K') \) then it follows from Prop. 14 (Rule R5) that \( (o \; pr' \; o') \in M_{K'} \). Thus, \( (o \; pr' \; o') \notin P_{K'} \), which is impossible.

\( \leftarrow \) Assume that (i) \( o \in URI \) and valid\( (o, \; pr', \; o', \; K') \), (ii) for all \( pr \in Pr_K \) s.t. \( pr' \leq_{PR} pr \), it holds that \( (o \; pr' \; o') \in (C_i(K') \cup P_K) \), and (iii) \( (o \; pr' \; o') \notin C_i(K') \). We will show that \( (o \; pr' \; o') \in P_{K'} \). It follows from Def. 5 that for all \( pr \in Pr_K \) s.t. \( pr' \leq_{PR} pr \), it holds that \( (o \; pr' \; o') \in (B_K \setminus M_K) \) or \( (o \; pr' \; o') \in C_i(K') \). It follows that \( (o \; pr' \; o') \notin M_K \), (note that Rule R2 and Rule R3 of Prop. 5 and Rule R5 of Prop. 14 do not apply).

It holds \( o \in Inst_{K'} \cap URI, \; o' \in Inst_{K'}, \) and \( pr \in Pr_K \). Thus, \( (o \; pr' \; o') \in B_{K'} \). Since \( (o \; pr' \; o') \notin C_i(K') \), it follows from Def. 5 that \( (o \; pr' \; o') \in P_{K'} \). \( \square \)

**Prop. 17** Let \( K = (S_K, I_K) \) be a KB and let \( S_K \) be the new schema version such that \( K' = (S_{K'}, I_K) \) is a (valid) KB. Then, \( P_{K'} = Produce\_Possibilities_{NBC}(K, P_K, S_{K'}) \).

**Proof:** Initially, \( P_{K, Add} = \emptyset \) and \( P_{K, Del} = \emptyset \). For each new class \( c' \in NC \), Algorithm \( Produce\_Possibilities_{NBC}(K, P_K, S_{K'}) \), inserts to \( P_{K, Add} \) exactly the class instance triples indicated by Prop. 15.

For each old class \( c_1 \in C_K \), if it exists \( c_2 \in C_K \) s.t. \( c_1 \leq_{G} c_2 \) and \( (o \; type \; c_2 \) \notin \( (P_K \cup C_i(K')) \) then all class instance triples \( (o \; type \; c_1) \in P_K \) are added to \( P_{K, Del} \). Note that \( (o \; type \; c_2) \notin (C_i(K) \cup P_K) \) or \( ((o \; type \; c_2) \in C_i(K) \) and \( (o \; type \; c_2) \notin C_i(K') \)). Since \( o \in Inst_{K} \cap URI \) and \( c \in C_K \), it follows that \( (o \; type \; c_2) \in B_K \). If \( (o \; type \; c_2) \notin (C_i(K) \cup P_K) \) then it follows from Def. 5 that \( (o \; type \; c_2) \in M_K \). Therefore, since \( (o \; type \; c_2) \notin C_i(K') \), it follows that \( (o \; type \; c_1) \in M_{K'} \) (see Rule R1 of Prop. 5). If \( (o \; type \; c_2) \in C_i(K) \) and \( (o \; type \; c_2) \notin C_i(K') \) then from Prop. 14 (Rule R4) it follows that \( (o \; type \; c_1) \in M_{K'} \).

Additionally, it adds to \( P_{K, Del} \), all class instance triples \( (o \; type \; c) \in P_K \), where \( c \notin C_{K'} \), because these instance triples do not belong to \( B_{K'} \).
Similarly, for each new property pr' ∈ NP, Algorithm \textit{Produce Possibilities}_{NBC}(K, P_K, S_{K'}) inserts to P_{K,Add} exactly the property instance triples indicated by Prop. 16. For each old property pr_1 ∈ Pr_K, if it exists pr_2 ∈ Pr_K s.t. pr_1 ≤_{pr} pr_2, and (o pr_2 o') ∉ (P_K ∪ C_i(K')) then all class instance triples (o pr_1 o') ∈ P_K are added to P_{K,Del}. Note that (o pr_2 o') ∉ (C_i(K) ∪ P_K) or ((o pr_2 o') ∈ C_i(K) and (o pr_2 o') ∉ C_i(K')). Obviously, (o pr_2 o') ∈ B_K. If (o pr_2 o') ∉ (C_i(K) ∪ P_K) then it follows from Def. 5 that (o pr_2 o') ∈ M_K. Therefore, since (o pr_2 o') ∉ C_i(K'), it follows that (o pr_1 o') ∈ M_{K'} (see Rule R2 of Prop. 5). If (o pr_2 o') ∈ C_i(K) and (o pr_2 o') ∉ C_i(K') then it follows from Prop. 14 (Rule R5) that (o pr_1 o') ∈ M_{K'}.

Additionally, it adds to P_{K,Del} all property instance triples (o pr o') ∈ P_K, where pr ∉ Pr_{K'} or \textit{valid}(o, pr, o', K'). This is because, in the first case they do not belong to B_{K'} and, in the second case, they belong to M_{K'} (see Rule R3 of Prop. 5).

All instance triples in P_K ∩ C_i(K') are moved to P_{K,Del}. Finally, P_{K'} = (P_K \setminus P_{K,Del}) ∪ P_{K,Add}. \hfill \Box

\textbf{Prop. 18} The time complexity of Algorithm 7 is \(O(|\text{Inst}_K|^2 * S^2 * (S^2 + |P_K|))\), where \(S = \max(|K|, |K'|)\).

\textbf{Proof:} In the proof of Prop. 10, we have shown that if \(K\) is a KB then the size complexity of \(C(K)\) is in \(O(|K|^2)\) and (ii) the time complexity of computing \(C(K)\) is in \(O(|K|^4)\).

Here, we will provide the complexity of the parts that Algorithm 7 differs from Algorithm 2. The complexity of the parts that Algorithm 7 is the same with Algorithm 2 is provided in the proof of Prop. 10, where we replace \(|K|\) and \(|K'|\) by \(S\). The time complexity of lines (9-10) is \(|P_K| * S^2\). The time complexity of lines (16-17) is \(|P_K| * S^2\). The time complexity to compute \(C(K)\) and \(C(K')\) is \(O(S^4)\). Thus, the total complexity of Algorithm 7 is \(O(|\text{Inst}_K|^2 * S^2 * (S^2 + |P_K|))\). \hfill \Box

\textbf{Prop. 19} Let \(K_1 = (S_1, I_1)\) be a KB and let \(S_2, \ldots, S_n\) be a sequence of backwards compatible schema versions, resulting to KBs \(K_i = (S_i, I_1)\), for \(i = 2, \ldots, n\). Now consider the one-step migration of \(I_1\) from the first \((S_1)\) to the last \((S_n)\) schema: \((C_1, M_1, P_1) \leadsto (C_n, M_n, P_n)\). Additionally, consider the sequential migrations scenario, where: \((C_1, M_1, P_1) \leadsto (C_2, M_2, P_2) \leadsto \ldots \leadsto (C_n, M_n, P_n)\).
\((C'_2, M'_2, P'_2) \sim \ldots \sim (C'_n, M'_n, P'_n)\). If (a) for all \(c_1, c_2 \in C_{K_1}\), it holds that \(c_1 \leq_{cl} c_2\) in \(K_1\) iff \(c_1 \leq^*_{cl} c_2\) in \(K_n\), and (b) for all \(pr_1, pr_2 \in Pr_{K_1}\), it holds that \(pr_1 \leq^*_{pr} pr_2\) in \(K_1\) iff \(pr_1 \leq_{pr} pr_2\) in \(K_n\), then \((C_n, M_n, P_n) = (C'_n, M'_n, P'_n)\).

**Proof:** Obviously, \(C_n = C'_n\). Let \((o \ type \ c) \in M'_n\). Then, there is a class \(c' \in C_{K_1} \cap C_{K_n}\) s.t. \(c \leq^*_{cl} c'\) in \(K_n\) and \((o \ type \ c') \in M_1 \cap M'_1\). Assume that \((o \ type \ c) \not\in M_n\). Then, \((o \ type \ c) \in C_n \cup P_n\). If \((o \ type \ c) \in C_n\) then this is impossible because \(C_n = C'_n\). If \((o \ type \ c) \in P_n\) then \((o \ type \ c') \in P_n \cup C_n\), due to Lemma 1(1). If \((o \ type \ c') \in P_n\) then this is also impossible, because \((o \ type \ c') \in M_1\) and due to Postulate \(\Pi_2\) of Def. 7. If \((o \ type \ c') \in C_n\) then \(c' \in C_{K_1} \cap C_{K_n}\). It holds that it exists \(c_1 \in C_{K_1}\) s.t. \(c_1 \leq^*_{cl} c'\) in \(K_n\) and (a) there is \((o \ property \ o') \in I_1\) and \(\text{domain}(pr) \leq^*_{pr} c_1\) in \(K_1\), or (b) there is \((o' \ property \ o) \in I_1\) and \(\text{range}(pr) \leq_{cl} c_1\) in \(K_1\), or (c) there is \((o \ type \ c_1) \in I_1\). Since \(c_1, c' \in C_{K_1} \cap C_{K_n}\), it follows from the assumption of Prop. 19(a) that \(c_1 \leq^*_{cl} c'\) in \(K_1\). Thus, \((o \ type \ c') \in C_1\), which is impossible since \((o \ type \ c') \in M_1\). Thus, \((o \ type \ c) \in M_n\).

Let \((o \ type \ c) \in M_n\). Then, there is a class \(c' \in C_{K_1} \cap C_{K_n}\) s.t. \(c \leq^*_{cl} c'\) in \(K_n\) and \((o \ type \ c') \in M_1 \cap M'_1\). Assume that \((o \ type \ c) \not\in M'_n\). Then, \((o \ type \ c) \in C'_n \cup P'_n\). If \((o \ type \ c) \in C'_n\) then this is impossible because \(C_n = C'_n\). If \((o \ type \ c) \in P'_n\) then \((o \ type \ c') \in P'_n \cup C'_n\), due to Lemma 1(1). If \((o \ type \ c') \in P'_n\), this is impossible due to the fact \((o \ type \ c') \in M_1\) and postulate \(\Pi_2\). If \((o \ type \ c') \in C'_n\) then \(c' \in C_{K_1} \cap C_{K_n}\). It holds that it exists \(c_1 \in C_{K_1}\) s.t. \(c_1 \leq^*_{cl} c'\) in \(K_n\) and (a) there is \((o \ property \ o') \in I_1\) and \(\text{domain}(pr) \leq^*_{pr} c_1\) in \(K_1\), or (b) there is \((o' \ property \ o) \in I_1\) and \(\text{range}(pr) \leq_{cl} c_1\) in \(K_1\), or (c) there is \((o \ type \ c_1) \in I_1\). Since \(c_1, c' \in C_{K_1} \cap C_{K_n}\), it follows from the assumption of Prop. 19(a) that \(c_1 \leq^*_{cl} c'\) in \(K_1\). Thus, \((o \ type \ c') \in C_1\), which is impossible since \((o \ type \ c') \in M_1\). Thus, \((o \ type \ c) \in M'_n\).

Let \((o \ property \ o') \in M'_n\). Then, there is a property \(pr' \in Pr_{K_1} \cap Pr_{K_n}\) s.t. \(pr \leq^*_{pr} pr'\) in \(K_1\) and \((o \ property \ o') \in M_1 \cap M'_1\). Assume that \((o \ property \ o') \not\in M_n\). Then, \((o \ property \ o') \in C_n \cup P_n\). If \((o \ property \ o') \in C_n\) then this is impossible because \(C_n = C'_n\). If \((o \ property \ o') \in P_n\) then \((o \ property \ o') \in P_n \cup C_n\), due to Lemma 1(2). If \((o \ property \ o') \in P_n\) then this is also impossible, because \((o \ property \ o') \in M_1\) and due to Postulate \(\Pi_2\) of Def. 7. If \((o \ property \ o') \in C_n\) then \(pr' \in Pr_{K_1} \cap Pr_{K_n}\). It holds that it exists \(pr_1 \in Pr_{K_1}\) s.t. \(pr_1 \leq_{pr} pr'\) in \(K_n\) and \((o \ property \ o') \in I_1\). Since \(pr_1, pr' \in Pr_{K_1} \cap Pr_{K_n}\), it follows from the assumption of Prop. 19(b)
that \( pr_1 \leq_{cl} pr' \) in \( K_1 \). Thus, \((o \ pr' \ o') \in C_1\), which is impossible since \((o \ pr' \ o') \in M_1\). Thus, \((o \ pr \ o') \in M_n\).

Let \((o \ pr \ o') \in M_n\). Then, there is a property \( pr' \in Pr_{K_1} \cap Pr_{K_n} \) s.t. \( pr \leq_{pr} pr' \) in \( K_n \) and \((o \ pr' \ o') \in M_1 \cap M_n\). Assume that \((o \ pr' \ o') \notin M_n'\). Then, \((o \ pr \ o') \in C_n' \cup P_n'\).

If \((o \ pr \ o') \in C_n'\), then this is impossible because \( C_n = C_n'\). If \((o \ pr \ o') \in P_n'\), then \((o \ pr \ o') \in P_n' \cup C_n',\) due to Lemma 1(2). If \((o \ pr' \ o') \in P_n'\), this is impossible due to the fact \((o \ pr \ o') \in M_1\) and postulate \( \Pi_2\). If \((o \ pr' \ o') \in C_n'\) then \( pr' \in Pr_{K_1} \cap Pr_{K_n}\).

It holds that it exists \( pr_1 \in Pr_{K_1} \) s.t. \( pr_1 \leq_{pr} pr' \) in \( K_n \) and \((o \ pr_1 \ o') \in I_1\). Since \( pr_1, pr' \in Pr_{K_1} \cap Pr_{K_n}\), it follows from the assumption of Prop. 19(b) that \( pr_1 \leq_{cl} pr' \) in \( K_1\). Thus, \((o \ pr' \ o') \in C_1\), which is impossible since \((o \ pr' \ o') \in M_1\). Thus, \((o \ pr \ o') \in M_n'\).

Therefore, \( M_n = M_n'\). The fact that \( P_n = P_n'\) follows immediately from the definition of X-partition (Def. 5). \(\square\)
Appendix B: List of Symbols

In this Appendix, we provide the list of symbols used in this thesis.
The set of added possible instance triples in $\mathcal{C}(K)$

$\mathcal{C}(K)$

The set of classes of $\mathcal{C}(K)$

$P^r_K$

The set of properties of $\mathcal{C}(K)$

$\leq_{cl}$

The subClassOf relation between classes in $\mathcal{C}(K)$

$\leq_{pr}$

The subPropertyOf relation between properties in $P^r_K$

$R e^K$

The resources of a KB $K$

$\text{Inst}_K$

The instances of a KB $K$

$\text{inst}_K(c)$

The instances of a class $c$ of a KB $K$

$B^*_K$

The set of cartesian instance triples of a KB $K$

$S^*_K$

The set of schema triples of a KB $K$

$I^*_K$

The set of instance triples of a KB $K$

$\mathcal{C}(I^*_K)$

The instance triples of the closure of a KB $K$

$\text{valid}(o, pr, o', K)$

A valid property instance triple of a KB $K$

$\text{Invalid}(K)$

The set of invalid property instance triples of a KB $K$

$\text{SubTriples}(o)$

The set of subtriples of a class instance triple $(o, c)$ of a KB $K$

$\text{SubTriples}(pr)$

The set of subtriples of a property instance triple $(o, pr, o')$ of a KB $K$

$\text{SubTriples}(A)$

The set of subtriples of a set of instance triples $A$ of a KB $K$

$M^*_K$

The set of negative(false) instance triples of a KB $K$

$P^*_K$

The set of all certain classes of an instance $c$

$P^*_K_{Add}$

The set of added possible instance triples in $P^*_K$

$P^*_K_{Del}$

The set of deleted possible instance triples from $P^*_K$

$\text{posTriples}_\ast(o)$

The set of possible class instance triples of an instance $o$

$\text{posTriples}_\ast^{pr}(o)$

The set of possible property instance triples of an instance $o$ with property $pr$

$\text{SupTriples}(o)$

The set of supertuples of a class instance triple $(o, c)$ of a KB $K$

$\text{SupTriples}(pr)$

The set of supertuples of a property instance triple $(o, pr, o')$ of a KB $K$

$\text{SupTriples}(A)$

The set of supertuples of a set of instance triples $A$ of a KB $K$

$K^o_{up}$

The updated certain part of an eKB

$K^o_{pr}$

The updated possible part of an eKB

$dist_{cl}(c \rightarrow c')$

The length of the shortest path from a class $c$ to a class $c'$

$\text{distClass}(o, c)$

The shortest distance of $c$ from one of the certain classes of $o$

$\text{dist}_{pr}(pr \rightarrow pr')$

The length of the shortest path from a property $pr$ to a property $pr'$

$\text{distProperty}(o, pr, o')$

The shortest distance of $pr$ from one of the certain properties of $(o, o')$

$\text{Valid}(K)$

The set of valid property instance triples of a KB $K$

$\text{posProp}$

The set of composite possibilities

$\text{posProp}_{ext}$

The set of extended possibilities including atomic and composite possibilities

$\mathcal{C}(o)$

The set of all certain properties of an instance $o$

$\text{posCl}(o)$

The set of all possible properties of an instance $o$

$\text{comp}_{K}$

The compact version of $P^*_K$

$P_{\text{compact}}(o)$

The intervals regarding an instance $o$

$\text{len}_{\text{compact}}(o)$

The number of intervals regarding an instance $o$

$\text{degree}(int)$

The number of classes/properties that occur in an interval $int$

$t_{\text{search}}(o)$

The time for locating an instance $o$ in the list of lexicographically ordered instances

$t_{\text{search}}(o, o')$

The time for locating a pair of instances $(o, o')$ in the list of lexicographically ordered pairs of instances

$t_{\text{subCheck}}^*$

The time for checking a subsumption relationship between classes

$t_{\text{subCheck}}^\ast$

The time for checking a subsumption relationship between properties

Table 1: Symbols and Description