knowledgeweb
realizing the semantic web

D2.3.9 Theoretical Aspects for Ontology Lifecycle

Coordinator: Vít Nováček[1]

Zhisheng Huang[2], Alessandro Artale[3], Norman Foo[9], Enrico Franconi[3], Tommie Meyer[10], Mathieu d'Aquin[11], Jean Lieber[4], Amedeo Napoli[4], Giorgos Flouris[5], Jeff Z. Pan[6], Dimitris Plexousakis[5], Holger Wache[7], Heiner Stuckenschmidt[8], Siegfried Handschuh[1]

[1]**DERI, National University of Ireland, Galway;** [2]**Vrije Universiteit, Amsterdam, Netherlands;** [3]**Free University of Bozen-Bolzano, Italy;** [4]**LORIA, France;** [5]**FORTH ICS, Greece;** [6]**University of Aberdeen, Aberdeen, U.K.;** [7]**FHNW, Switzerland;** [8]**University of Mannheim, Germany;** [9]**NICTA, Australia;** [10]**Meraka Inst., South Africa;** [11]**KMI, London, U.K.**

Abstract.
Deliverable D2.3.9 (WP2.3) presents a study on theoretical aspects of ontology lifecycle and dynamic maintenance. Several crucial topics are analysed, ranging from logical groundwork of ontology dynamics based on belief-change theory, through semantics of ontology diffs, to multi-version reasoning. Building on the theoretical studies, basic practical guidelines and combined approach to multi-version ontology reasoning are discussed in the report, too. This is meant to provide a tangible binding between the theory and interests of practitioners.

Keyword list: ontology, ontology dynamics, ontology lifecycle, ontology maintenance, belief-change, semantics of ontology diffs, multi-version reasoning

Document Identifier	KWEB/2007/D2.3.9
Project	KWEB EU-IST-2004-507482
Version	v1.0
Date	November 14, 2007
State	final
Distribution	public

Knowledge Web Consortium

This document is part of a research project funded by the IST Programme of the Commission of the European Communities as project number IST-2004-507482.

University of Innsbruck (UIBK) - Coordinator
Institute of Computer Science
Technikerstrasse 13
A-6020 Innsbruck
Austria
Contact person: Dieter Fensel
E-mail address: dieter.fensel@uibk.ac.at

École Polytechnique Fédérale de Lausanne (EPFL)
Computer Science Department
Swiss Federal Institute of Technology
IN (Ecublens), CH-1015 Lausanne
Switzerland
Contact person: Boi Faltings
E-mail address: boi.faltings@epfl.ch

France Telecom (FT)
4 Rue du Clos Courtel
35512 Cesson Sévigné
France. PO Box 91226
Contact person : Alain Leger
E-mail address: alain.leger@rd.francetelecom.com

Freie Universität Berlin (FU Berlin)
Takustrasse 9
14195 Berlin
Germany
Contact person: Robert Tolksdorf
E-mail address: tolk@inf.fu-berlin.de

Free University of Bozen-Bolzano (FUB)
Piazza Domenicani 3
39100 Bolzano
Italy
Contact person: Enrico Franconi
E-mail address: franconi@inf.unibz.it

Institut National de Recherche en Informatique et en Automatique (INRIA)
ZIRST - 655 avenue de l'Europe -
Montbonnot Saint Martin
38334 Saint-Ismier
France
Contact person: Jérôme Euzenat
E-mail address: Jerome.Euzenat@inrialpes.fr

Centre for Research and Technology Hellas / Informatics and Telematics Institute (ITI-CERTH)
1st km Thermi - Panorama road
57001 Thermi-Thessaloniki
Greece. Po Box 361
Contact person: Michael G. Strintzis
E-mail address: strintzi@iti.gr

Learning Lab Lower Saxony (L3S)
Expo Plaza 1
30539 Hannover
Germany
Contact person: Wolfgang Nejdl
E-mail address: nejdl@learninglab.de

National University of Ireland Galway (NUIG)
National University of Ireland
Science and Technology Building
University Road
Galway
Ireland
Contact person: Christoph Bussler
E-mail address: chris.bussler@deri.ie

The Open University (OU)
Knowledge Media Institute
The Open University
Milton Keynes, MK7 6AA
United Kingdom
Contact person: Enrico Motta
E-mail address: e.motta@open.ac.uk

Universidad Politécnica de Madrid (UPM)
Campus de Montegancedo sn
28660 Boadilla del Monte
Spain
Contact person: Asunción Gómez Pérez
E-mail address: asun@fi.upm.es

University of Karlsruhe (UKARL)
Institut für Angewandte Informatik und Formale Beschreibungsverfahren - AIFB
Universität Karlsruhe
D-76128 Karlsruhe
Germany
Contact person: Rudi Studer
E-mail address: studer@aifb.uni-karlsruhe.de

University of Liverpool (UniLiv)
Chadwick Building, Peach Street
L697ZF Liverpool
United Kingdom
Contact person: Michael Wooldridge
E-mail address: M.J.Wooldridge@csc.liv.ac.uk

University of Manchester (UoM)
Room 2.32. Kilburn Building, Department of Computer Science, University of Manchester, Oxford Road
Manchester, M13 9PL
United Kingdom
Contact person: Carole Goble
E-mail address: carole@cs.man.ac.uk

University of Sheffield (USFD)
Regent Court, 211 Portobello street
S14DP Sheffield
United Kingdom
Contact person: Hamish Cunningham
E-mail address: hamish@dcs.shef.ac.uk

University of Trento (UniTn)
Via Sommarive 14
38050 Trento
Italy
Contact person: Fausto Giunchiglia
E-mail address: fausto@dit.unitn.it

Vrije Universiteit Amsterdam (VUA)
De Boelelaan 1081a
1081HV. Amsterdam
The Netherlands
Contact person: Frank van Harmelen
E-mail address: Frank.van.Harmelen@cs.vu.nl

Vrije Universiteit Brussel (VUB)
Pleinlaan 2, Building G10
1050 Brussels
Belgium
Contact person: Robert Meersman
E-mail address: robert.meersman@vub.ac.be

University of Aberdeen (UNIABDN)
Kings College
AB24 3FX Aberdeen
United Kingdom
Contact person: Jeff Pan
E-mail address: jpan@csd.abdn.ac.uk

Work package participants

The following partners have taken an active part in the work leading to the elaboration of this document, even if they might not have directly contributed to writing parts of this document:

National University of Ireland Galway
Institut National de Recherche en Informatique et en Automatique
Vrije Universiteit Amsterdam
University of Aberdeen
Centre for Research and Technology Hellas / Informatics and Telematics Institute (ITI-CERTH)
Free University of Bozen-Bolzano (FUB)
KMI, Open University, London

Changes

Version	Date	Author	Changes
0.1	01.10.07	Vít Nováček	draft created
0.2	20.10.07	Vít Nováček	draft updated (using Free University of Amsterdam et al. contributions for Chapter 2 and Chapter 4)
0.3	26.10.07	Vít Nováček	draft updated (major rewrite of Chapter 1 and Chapter 6)
0.4	09.11.07	Vít Nováček	Chapter 3 completed, based on Free University of Bozen-Bolzano contribution
0.5	13.11.07	Vít Nováček	Chapter 5 completed, based on LORIA/INRIA contribution
1.0	14.11.07	Vít Nováček	final editing, incorporation of the University of Karlsruhe QC

Executive Summary

We present a report on several theoretical issues underlying the ontology dynamics and ontology maintenance process. Ontology evolution topics have been widely discussed within the Semantic Web community recently, both from practical and theoretical points of view. Our motivation is to give an overview of selected important theoretical problems related to ontology dynamics and discuss possible approaches to their solutions, stemming from state of the art research. Though not directly related, this report complements the dynamic ontology lifecycle methodology and partial implementation, as described in deliverables D2.3.8v1 [NHL$^+$06] and D2.3.8v2 [NLHZ07].

In this report, we address namely the following areas:

1. **Logical Groundwork for Changes in Ontologies** – study on ontology change operations well-founded using belief-change theory
2. **Semantics of Ontology Diffs** – formal definition of an ontology diff, study on the relations between syntactic and semantic level of ontology diffs in RDFS
3. **Reasoning with Versioned Ontologies** – study on realisations of inference among multiple versions of an ontology; two possible approaches analysed:
   - multi-version inference using linear temporal logics among sequence of ontology versions (the reasoning about particular ontologies in the version sequence is realised using a "classical" ontology inference engine, whereas the reasoning across the whole sequence is done by model checking)
   - multi-version inference using the C-OWL formalism among set of ontology versions connected by so called bridge rules (inference is based on the Distributed Description Logics formalism, utilising localised Description Logics reasoning for particular ontology versions and global ontology mappings, i.e. the bridge rules, for the whole version sequence)

Building on the theoretical study of the enumerated topics, we also discuss basic practical consequences concerning the dynamic ontology maintenance process. We also discuss possible combination of the two presented alternatives of multi-version reasoning and provide general suggestions on implementation and application of the combined approach. The main expected contribution consists of the profound and coherent analysis of the above theoretical issues, complemented by identification of several important general connections to practical implementations tackling ontology dynamics. In this respect, we also give basic suggestions on incorporation of the theoretical results into practical scenarios.

Contents

Chapter 1

Introduction

The purpose of this deliverable is to introduce several theoretical aspects relevant for dynamics in the ontology lifecycle. We focus mainly on two issues arising when dealing with ontologies dynamically changing in time – **formalisations of ontology change** and **reasoning across multiple versions** (of a changing ontology).

The presented content is meant to complement the practical features of dynamic ontology lifecycle described in [NHL+06, NLHZ07], where we concentrate on certain practical parts of the ontology evolution process. More specifically, the delivered analysis of theoretical aspects provides a support for efficient and well-founded maintenance and exploitation of changing ontologies

1.1 High-Level Overview and Motivation

We do not give exhaustive analysis of every imaginable theoretical issue related to ontology dynamics. More specifically, we omit a more detailed analysis of inconsistency handling and resolution issues (see for instance [HvHH+05, HvHtT05]), which we consider to be out of scope here, though definitely related to ontology lifecycle. We pay attention to the two general problems mentioned above. They are elaborated in more detail according to the following:

1. formalisation of ontology change:
    - Chapter 2 – proper elaboration of the relation between inconsistency and negation, building on the AGM theory [AGM85], and consecutive formulation of formal postulates for ontology change (contraction and revision operations)
    - Chapter 3 – formalisation of the ontology diff notion, specification of its semantic properties, study of the relation between the syntactic and semantic levels of RDFS ontology diffs
2. multi-version reasoning:

- Chapter 4 – study on a temporal logics inference (i.e., model checking) on the top of "classical" ontology reasoning, definition of a respective query language for multi-version reasoning
- Chapter 5 – adaptation of C-OWL [BGvH+03] formalism for modelling of and reasoning with contextualised ontologies in OWL [BvHH+04] in order to allow inference across sequences of ontology versions, linked by so called bridge rules

Thus, the report provides information on: (1), how to formally grasp ontology change; (2), how to benefit from this formalisation in order to facilitate dynamic ontology maintenance; (3), how to exploit the versioned changing ontologies with multi-version reasoning. Some basic guidelines in this respect are elaborated in the concluding Chapter 6, where we relate the rather theoretical content to the practical implementation of the dynamic ontology lifecycle features.

The first motivation and aim is to provide the implementers and users of dynamic ontology lifecycle applications with principles of consistent, efficient and well-founded ontology maintenance. The second aim is to present possible alternatives of multi-version inference in order to offer means for reasoning with changing ontologies. Fulfilment of both these general aims forms also the main contribution of this report.

Note that our overall motivations are supported by the community demands analysed in a survey aimed at several features of ontology dynamics (results reported in [NLHZ07])). More detailed motivations relevant to the particular topics are given in the introductory parts of the respective chapters.

1.2 Related Work

The general topic of ontology evolution and change has been studied for instance in [NK04, Sto04]. These approaches cover the changing ontology maintenance mostly from the practical point of view, supporting appropriate applications dealing with changing ontologies. In our report, we aim to provide a better understanding of what the change actually is and what the consequences of its introduction can be.

Ontology versioning theory, methodology and implementations supporting ontology change are covered by [KFKO02, KF01, VG06]. These works provide means for management of changing ontologies, building on the well-known principles from software and database schema maintenance. However, the support for actual exploitation of resulting ontology version repositories is lacking. In this document, we offer solutions how to tackle this exploitation, namely by means of multi-version inference.

A framework for changing ontologies, specifically for handling possible inconsistencies is given in [HvHH+05]. [HvHtT05] analyses possibilities of reasoning with an inconsistent ontology. These approaches can be seen as complementary to the work presented here, since we deliberately disregard analysis of inconsistency detection and resolution in

favour of presentation of formalisms that can possibly minimise inconsistency introduction to large extent.

Furthermore, relatively representative sample of the most recent research trends in the theoretical support of ontology dynamics can be found through the web-site of the IWOD'07 workhop (see `http://kmi.open.ac.uk/events/iwod/`). The NEON project has focused on a special issues of dynamics among *networked* ontologies, however, the state of the respective research has been still rather in its initial stage by the date of publication of this report, largely in line with the directions of the works cited above.

Similarly to the motivations, more detailed overview of a specific related work is given in the particular chapters.

1.3 Position within the Project

Concerning the dynamics Knowledge Web work-package (WP 2.3), the work presented here has direct relation to dynamic ontology lifecycle [NHL+06, NLHZ07]. This relation holds namely for Chapters 2 and 3 that present theoretical analysis underlying well-founded, optimal and efficient dynamic ontology maintenance. Chapters 4 and 5, dealing with reasoning across multiple versions of an ontology, are primarily related to the ontology versioning topics [VEK+05, VKZ+05].

There is no direct relation to other work-packages in the Knowledge Web project, since the presented content is very explicitly focused on the basic theoretical issues closely associated with the ontology dynamics only. Indirectly, the proposal of C-OWL based multi-version reasoning in Chapter 5 can be related to the heterogeneity Knowledge Web work-package (WP 2.2), since C-OWL may be used as an ontology mapping language, too (among other things).

Chapter 2

Logical Groundwork for Changes in Ontologies

by GIORGOS FLOURIS, ZHISHENG HUANG, JEFF Z. PAN, DIMITRIS PLEXOUSAKIS, AND HOLGER WACHE[1]

This chapter introduces a logic-based framework for ontology change. The change management plays its role in the essential parts of the ontology lifecycle scenario (mainly *versioning* and *creation*), as presented in [NHL$^+$06]. Such a framework is therefore no doubt very relevant in this respect, however it is relatively difficult to directly apply its theoretical conclusions in practice. The logical framework described in this chapter formally underpins general changes in DL-based ontologies in terms of operators of *revision* (addition of new statements) and *contraction* (removal of current statements). These operators should desirably preserve the consistency of an ontology in practical application. Thus, requirements on the operators and the very notions related to the consistency interpretations (mainly negation) must be rigorously specified before implementing the respective operations within practical realisation of an ontology lifecycle. We present such an analysis here.

The ability to deal with inconsistency and to accommodate change is of utmost importance in real-world applications of Description Logic based ontological reasoning and management [BCM$^+$03b, HST00]. For example, one of the typical scenarios in deployed Semantic Web applications is ontology reuse, where users build their own ontologies from existing ones, rather than starting from scratch. After adding new axioms into an existing ontology, users may find that revised ontologies become inconsistent. A remedy for such a situation would require the removal of a minimal part of the ontology in order to make the resulting ontology consistent [HvHH$^+$05]. This type of change is usually required to meet some rationality postulates, similar to those in the AGM theory in the belief revision [AGM85]. Another example is reasoning with inconsistent ontologies [HvHtT05], where querying systems should return meaningful answers to queries on inconsistent ontologies. The latter suffers from ”entailment explosion” as any formula is a consequence of an inconsistent logical theory.

[1] Based on [FHP$^+$06], with minor modifications by Vít Nováček.

Addressing effectively the issues raised in these examples requires precise, formal definitions of inconsistency and negation. Unfortunately, DL-based ontology languages, such as OWL DL [PSHH04], do not provide enough expressive power to represent axiom negations. Furthermore, there is no single, well-accepted notion of inconsistency and negation in the Semantic Web community, due to the lack of a common and solid foundational framework. [SC03] proposed an approach to debug inconsistent ontologies, in which inconsistency is identified with the existence of unsatisfiable concepts. [HvHtT05] developed a framework of reasoning with inconsistent ontologies, in which inconsistency is given a classical first-order logic interpretation. In [HvHH$^+$05], the definition of axiom negation is merely mentioned in an example at a footnote, without proper discussion in the paper.

We propose a general framework accounting for inconsistency, negation and change by which we aim at providing a unique foundation of inconsistency and change processing for DL-based ontologies. We distinguish different levels of inconsistency and negation in DL-based ontologies, and investigate the relationship among the different notions. Accordingly, we lay the foundations of a formal theory of ontology change, based on a set of rationality postulates inspired by the AGM theory of belief change. Furthermore, we discuss how this proposed framework can provide a foundation for the tasks of ontology management and reasoning. Specifically, we show how a bridge connecting two main ontology change operations – revision and contraction – can be built under the proposed framework.

2.1 Preliminaries

Ontologies An *ontology* [UG96] typically consists of a hierarchical description of important concepts in a domain, along with descriptions of the properties of each concept, and constraints on these concepts and properties. In this chapter, following the W3C Web Ontology language OWL [PSHH04], we consider Description Logics (DLs) based ontologies. Description Logics are a family of class-based (concept-based) knowledge representation formalisms, equipped with well-defined model-theoretic semantics [BCM$^+$03b]. The $\mathcal{SHOIN}(\mathbf{D}^+)$ DL underpins OWL DL, the key sub-language of OWL. The relation between $\mathcal{SHOIN}(\mathbf{D}^+)$ DL and OWL DL – mainly concerning reduction of OWL entailment to DL satisfiability – is described in detail in [HPS03].

Let $\mathcal{K}$ be a Description Logic, C, D $\mathcal{K}$-concepts, R, S $\mathcal{K}$-roles, and $\mathtt{a}, \mathtt{b}$ individuals. An *interpretation* (written as $\mathcal{I}$) of an ontology consists of a *domain* $\Delta^{\mathcal{I}}$ (a nonempty set), and an *interpretation function* (written as $\cdot^{\mathcal{I}}$), which maps each individual name $\mathtt{a}$ to an element $\mathtt{a}^{\mathcal{I}} \in \Delta^{\mathcal{I}}$, each concept name CN to a subset $CN^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$ of the domain and each role name RN to a binary relation $RN^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$. The interpretation function can be extended to give semantics to $\mathcal{K}$-concepts and $\mathcal{K}$-roles, which are concepts and role descriptions built by $\mathcal{K}$-constructors. Example concept constructors of $\mathcal{SHOIN}(\mathbf{D}^+)$ are $\neg C, C \sqcap D, C \sqcup D, \exists R.C, \forall R.C, \geqslant nR, \leqslant nR$ and $\{\mathtt{a}\}$ (where n is a natural number).

A $\mathcal{K}$-ontology (or simply ontology) O is a finite set of axioms of the following forms:[2] concept inclusion axioms $C \sqsubseteq D$, transitivity (abstract) role axioms $\mathsf{Trans}(R)$, role inclusion axioms $R \sqsubseteq S$, concept assertions $C(\mathtt{a})$, role assertions $R(\mathtt{a}, \mathtt{b})$ and individual (in)equalities $\mathtt{a} \approx \mathtt{b}$ ($\mathtt{a} \not\approx \mathtt{b}$, respectively). In an ontology, we use *TBox* (*RBox*, *ABox*) to refer to the set of concept (role, individual, respectively) axioms. An interpretation $\mathcal{I}$ satisfies the concept inclusion axiom $C \sqsubseteq D$ if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$. Due to the limitation of space, the reader is referred to [BCM$^+$03b] for more details of the semantics of DL constructors and axioms. An interpretation $\mathcal{I}$ satisfies an ontology O iff $\mathcal{I}$ satisfies all its axioms. An ontology O is *consistent* iff it has an interpretation. A concept C is *satisfiable* w.r.t. O iff there exists an interpretation $\mathcal{I}$ of O s.t. $C^{\mathcal{I}} \neq \emptyset$. A concept C is subsumed by a concept D w.r.t. O iff, for every interpretation $\mathcal{I}$ of O, we have $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$. Given an axiom φ, an ontology O *entails* φ, written as $O \models \varphi$, iff, for all interpretations $\mathcal{I}$ of O, we have $\mathcal{I}$ satisfies φ. An ontology O_1 entails an ontology O_2, written as $O_1 \models O_2$, iff, for all interpretations $\mathcal{I}$ of O_1, we have $\mathcal{I}$ satisfies O_2.

Given a (monotonic) Description Logic $\mathcal{K}$, we consider a pair $\langle L, Cn\rangle$, where L is the set of possible $\mathcal{K}$-axioms and Cn is a consequence operator such that, given a $\mathcal{K}$-ontology O, $Cn(O) = \{\varphi \mid O \models \varphi\}$. In the rest of the chapter, we will use $\langle L, Cn\rangle$ (or $\langle L^{\mathcal{K}}, Cn\rangle$ when necessary) to refer to the Description Logic $\mathcal{K}$. $\langle L, Cn\rangle$ is a very general model introduced by Tarski in 1928; to guarantee rationality, Tarski required that Cn satisfies *iteration*, *inclusion* and *monotony*; see [Fuh91].

AGM Theory and its Variations The theory of Alchourrón, Gärdenfors and Makinson [AGM85] — the *AGM theory* — is probably the most influential work in the area of belief change. This theory sets the foundations for future research on belief change, by defining a set of widely accepted properties that any rational operators should satisfy.

More specifically, AGM studied 3 different operators, namely *expansion*, *revision* and *contraction*. Expansion is the addition of a sentence to a knowledge base (KB), without taking any special provisions for maintaining consistency; revision is similar, with the important difference that the result should be a consistent set of beliefs; contraction is required when one wishes to consistently remove a sentence from their beliefs instead of adding one. AGM introduced a set of postulates for revision and contraction that formally describe the properties that such an operator should satisfy (expansion was skipped, as it is trivial).

The AGM theory is based on the *coherence model*. In practice, this model states that both the explicitly represented knowledge and the implied knowledge are of equal value and should be considered when deciding the changes to be made upon the KB. In the context of ontologies, however, it seems more natural to use the *foundational model*, under which there is a clear distinction between the explicitly represented knowledge (i.e., the one contained in the KB) and the implicit one (i.e., knowledge implied by the explicitly represented one). Under this model, changes can be made in the explicit knowledge only; implicit knowledge can only be indirectly affected through changes in the explicit

[2] The kinds of role axioms that can appear in O depend on the expressiveness of $\mathcal{K}$.

knowledge.

The foundational model greatly restricts our options for a "proper" modification of knowledge. This fact was verified in [Fuh91], in which an attempt to define a foundational version of the AGM theory was made. There it was shown that, in the logics originally considered by AGM, no contraction operator can be defined that satisfies the foundational version of the AGM postulates.

A second problem related to the application of the AGM theory in the DL context is caused by the assumptions made by AGM in the formulation of their theory: even though the intuition behind the AGM postulates is independent of the logic used for the representation of the KB, the formulation of the postulates themselves is based on certain assumptions, disallowing their direct use in logics such as DLs [FPA04]. For example, well known DLs do not provide enough expressive power to represent negations of all the axioms. This fact is both a curse and a blessing. On the one hand, it implies that the AGM theory cannot be directly applied to DLs; on the other hand, if we could reformulate the AGM theory in a more general context, then the result of [Fuh91] might not be applicable in DLs, as they do not satisfy the AGM assumptions.

This problem was originally addressed in [FPA04], where the AGM theory (and postulates) were recast so as to be applicable in a wider class of logics, which includes DLs. That work studied the AGM theory under both the coherence and the foundational model, but was restricted to the operation of contraction only. It was shown that there are certain conditions under which a logic admits a contraction operator satisfying the AGM postulates in each of the two paradigms (coherence, foundational). Such logics were termed *AGM-compliant* and *base-AGM-compliant*, respectively.

In this chapter, we focus on the foundational model; we will show that the conditions introduced in [FPA04] for a base-AGM-compliant logic are too restrictive, overruling practically all interesting DLs. Following this observation, we propose a weakening of the AGM postulates which is applicable in our context (DLs under the foundational model) and present some ideas on the operation (and postulates) of revision and its interrelationship with contraction.

2.2 Inconsistency and Negation

Different notions of inconsistency in DLs have been used in the Semantic Web community, as we have discussed them in the introduction of this Chapter. We define different notions of inconsistency and examine their relations there. We start from the most primitive inconsistency, i.e., the unsatisfiability of a single concept.

Definition 1 (Unsatisfiable Concept) *A named concept C in the ontology O is unsatisfiable iff, for each interpretation $\mathcal{I}$ of O, $C^{\mathcal{I}} = \emptyset$.*

That would lead us to consider the kinds of ontologies with unsatisfiable concepts.

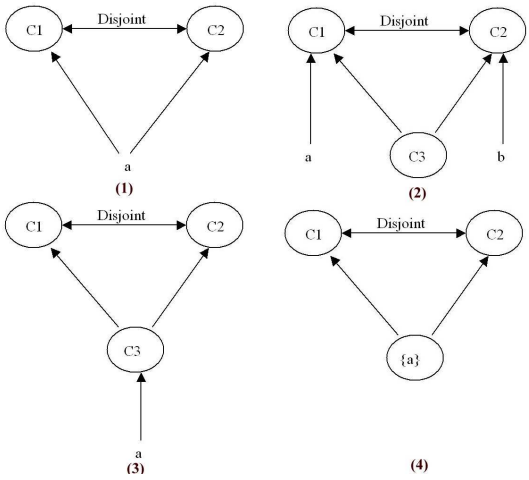

Figure 2.1: Examples of variant inconsistency and incoherence.

Definition 2 (Incoherent Ontology) *An ontology O is incoherent iff there exists an unsatisfiable named concept in O.*

The incoherence can be considered as a kind of the inconsistency in the TBox, i.e. the terminology part, of an ontology. An incoherent ontology has an incoherent TBox. However, the incoherence does not provide the classical sense of the inconsistency because there might exist a model for an incoherent ontology. Thus, we need the classical inconsistency for ontologies.

Definition 3 (Inconsistent Ontology) *An ontology is inconsistent iff it has no interpretation.*

We now briefly discuss the relationships of the two kinds of inconsistencies of ontologies. Firstly, an ontology is inconsistent does not necessarily imply that it is incoherent, and vice versa. There exist different combinations of the inconsistency and the incoherence. Figures 2.1 presents several examples to show the variants of inconsistency and incoherence. Figure2.1(1) is an example of inconsistent but coherent ontology, in which the two disjoint concepts $C1$ and $C2$ share an instance a. Figure2.1(2) is an example of consistent but incoherent ontology, in which the two disjoint concepts $C1$ and $C2$ share a sub-concept $C3$. Figure2.1(3) is an example of an inconsistent and incoherent ontology, in which the two disjoint concepts $C1$ and $C2$ share a sub-concept $C3$, which has an instance a. Figure2.1(4) is an example of inconsistent but coherent TBox, in which the two disjoint concepts $C1$ and $C2$ share a sub-concept which is a nominal $\{a\}$.

Secondly, coherence and consistency are somehow related. We can introduce a fresh individual i_C for each named concept C in an ontology O. Accordingly, an enhanced ontology $O^+ = O \cup \{C(i_C) \mid \text{for all named concepts } C \text{ in } O\}$ can be constructed by adding these individual axioms about these fresh individuals into the ontology. It is easy to see that the following propositions hold:

Proposition 2.2.1 *(a) Given an ontology O, if its enhanced ontology O^+ is consistent, then O is coherent.*
(b) Given a consistent ontology O, if O is coherent, then its enhanced ontology O^+ is consistent.

2.2.1 Axiom Negation in Ontologies

Negated axioms are closely related to inconsistencies and changes in ontologies. They are one of the main sources of ontology inconsistencies. For example, an ontology containing the mutually negated axioms $C(a), \neg C(a)$ is inconsistent. Furthermore, negated axioms are one of the keystones connecting the contraction and revision operators in the AGM theory, although unfortunately well known DL-based ontology languages do not provide enough expressive power to represent negations of all the axioms. Similar to the notion of inconsistency, the definition of the negation is different from an approach to another approach in the Semantic Web community [HvHH$^+$05, HvHtT05], as we have briefly discussed in the introduction of this chapter.

Based on the distinction between ontology consistency and coherence, in the following we propose two corresponding axiom negations.

Definition 4 (Consistency-Negation) *An axiom ψ is said to be a consistency-negation of an axiom ϕ, written $\psi = \neg\phi$, iff*
(i)(Inconsistency)
$\{\phi, \psi\}$ is inconsistent,
(ii) (Minimality)
There exist no other ψ' such that ψ' satisfies the condition (i) and $Cn(\{\psi'\}) \subset Cn(\{\psi\})$.

The inconsistency condition states the relationship between axiom negation and ontology inconsistency, which is based on the classical notion of negation. We introduce the minimality condition to make the negation minimal so that it would not include any unnecessary additional part. Note that this does not enforce a unique consistency-negation though. Similarly we have the following axiom negation which corresponds with incoherence.

Definition 5 (Coherence-Negation) *An axiom ψ is said to be a coherence-negation of an axiom ϕ, written $\psi = \sim\phi$, iff (i)(Incoherence) $\{\phi, \psi\}$ is incoherent,*
(ii) (Minimality) There exist no other ψ' such that ψ' satisfies the condition (i), and $Cn(\{\psi'\}) \subset Cn(\{\psi\})$.

Note that it is possible to extend our notion of negated axioms from a single axiom to a set of axioms, where a set of axioms represent the negation of another set of axioms. This extension goes beyond the scope of this chapter.

Example 1 *Let us consider the consistency negation and the coherence negation of an axiom $C \sqsubseteq D$, where C and D are named concepts.*

$$\neg(C \sqsubseteq D) = \exists(C \sqcap \neg D), \ \sim (C \sqsubseteq D) = C \sqsubseteq \neg D$$

where $\exists(C \sqcap \neg D)$ is an existence axiom [HPS03], which states there exists some instance of the concept $C \sqcap \neg D$. Note that, in any ontologies containing $C \sqsubseteq D$ and $C \sqsubseteq \neg D$, the concept C is unsatisfiable.

It should be noted that the minimality condition of the consistency-negation prevents the counter-intuitive property that any axiom ψ is qualified to be a consistent-negation of an inconsistent axiom ϕ (such as $C \sqsubseteq \neg C$). It is easy to see that its consistency-negation must be the tautology T because the tautology T is implied by any axiom ψ, i.e. $Cn(\emptyset) = Cn(\{T\}) \subseteq Cn(\{\psi\})$. Thus no other axioms can meet the minimality condition. For example, we have $\neg(\{a\} \sqsubseteq \bot) = T$; similarly, we have $\sim (D \sqsubseteq \bot) = T$.

In the following we will briefly discuss whether the proposed negations satisfy the following important properties:

1. *Existence*: It should exist in (almost) every DL.
2. *Classicality*: If the definition of negation is applied in a classical logic, it should coincide with the classical negation.
3. *Decidability*: The problem of checking whether or not an axiom is the negation of another axiom should be decidable.

Existence Definitions 4 and 5 improve the Existence property by giving up the restriction on double negations; i.e., an axiom ψ should be logically equivalent to the negation of the negation of ψ. Due to the limitation of space, here we only illustrate our point with some examples. For a DL $\langle L, Cn\rangle$ that does not provide concept existence axioms, we cannot use $\exists(C \sqcap \neg D)$ as a negation of $C \sqsubseteq D$; however, Definition 4 allows $C \sqcap \neg D(a)$ (where a is a fresh individual) as a consistency-negation of $C \sqsubseteq D$. For a DL $\langle L, Cn\rangle$ that does not provide any role constructors, we cannot use $\exists(R \sqcap \neg S)$ as a negation of the role inclusion $R \sqsubseteq S$; however, Definition 5 allows $C \sqsubseteq \exists R.\top \sqcap \forall S.\bot$ (where C is a fresh named concept) as a coherent-negation of $R \sqsubseteq S$.

Classicality The classical negation has the following intuitive properties:
(i) $Cn(\{\phi\}) \cap Cn(\{\neg\phi\}) \subseteq Cn(\emptyset)$ (only the tautology appears in both the consequences set of an axiom and its negation);
(ii) $Cn(\{\phi\} \cup \{\neg\phi\}) = L$ (the consequence set of the negation is the complement set of the consequence set of the axiom).

It can be shown that, under standard assumptions [FPA04], the properties are guaranteed by the consistency-negation.

Decidability Given a DL $\langle L, Cn\rangle$, let us first consider the consistency-negation. The checking of the inconsistency condition is indeed a knowledge base satisfiability problem

of $\langle L, Cn\rangle$. The minimality condition can be checked by trying to replace some sub-concepts (or sub-roles) with more general ones.

Proposition 2.2.2 *Given a DL $\langle L, Cn\rangle$, if the knowledge base satisfiability problem of $\langle L, Cn\rangle$ is decidable, then the consistency-negation checking in $\langle L, Cn\rangle$ is decidable.*

Proposition 2.2.3 *Given a DL $\langle L, Cn\rangle$, if the problem of concept satisfiability w.r.t. to a TBox in $\langle L, Cn\rangle$ is decidable, then the coherent-negation checking in $\langle L, Cn\rangle$ is decidable.*

2.3 Postulates for Ontology Change

In this section we propose certain postulates which describe two rational change operators for DL-based ontologies and even the use of the different kind of negation for revision. Our approach will be based on the AGM theory presented in Section 2.1. Note that proofs of the statements in this section can be found in [Flo06, FPA06].

2.3.1 Postulates for Contraction

The main result that motivates our quest for a new set of contraction postulates is summarized in the next lemma and its corollary. Note that, as mentioned above, we use $\langle L, Cn\rangle$ (or $\langle L^{\mathcal{K}}, Cn\rangle$ when necessary) to refer to the Description Logic $\mathcal{K}$.

Lemma 1 *For a DL $\langle L, Cn\rangle$, if there is an axiom $x \in L$ and a set of axioms $Y \subseteq L$ such that $Cn(\emptyset) \subset Cn(Y) \subset Cn(\{x\})$, then $\langle L, Cn\rangle$ is not base-AGM-compliant.*

Corollary 1 *Any DL that is at least as expressive as $\mathcal{FL}_0$ and whose alphabet allows at least two concept names and one role name is non-base-AGM-compliant.*

Corollary 1 practically overrules the use of the postulates that appeared in [FPA04] in the DL context. The reason for this failure is related to the so-called base recovery postulate (B-6). Here, we will propose a different set of postulates that satisfy the following:

1. *Existence*: For every monotonic DL $\langle L, Cn\rangle$, there is a contraction operator satisfying the proposed postulates.
2. *AGM Rationality*: Whenever possible (i.e., for base-AGM-compliant DLs), the proposed postulates allow exactly the same contraction operators as the AGM postulates do.

It turns out that the following set of postulates satisfies both goals:

(O-1) $O - X \subseteq O$.

(O-2) If $O \not\models X$, then $O - X = O$.

(O-3) If $\emptyset \not\models X$, then $O - X \not\models X$.

(O-4) If $X \cong Y$, then $O - X = O - Y$.

(O-5) If $Cn((O - X) \cup X) \subset Cn(Y \cup X)$ for some $Y \subseteq O$, then $Y \models X$ and $\emptyset \not\models X$.

The postulates (O-1)-(O-4) are equivalent reformulations of the postulates discussed in [FPA04], i.e. (B-2)-(B-5), respectively; postulate (B-1) from [FPA04] was ignored because it is trivial. These postulates follow the AGM intuition: contraction is an operation that is used to remove knowledge from an ontology, so the result should not contain any new, previously unknown, information (O-1); if the contracted axiom is not part of our original knowledge, nothing should be removed (O-2); but if it is, then contraction is supposed to return a new ontology such that the contracted expression is no longer explicitly asserted or entailed (O-3); finally, the result should be syntax-independent (O-4).

Postulates (O-1)-(O-4) fail to capture the *Principle of Minimal Change* [Gär92] which states that a contraction operator should remove as little information from the ontology as possible. This principle was originally captured by postulate (B-6) in [FPA04], while in our case it weakened to form (O-5). (B-6) states that a contraction operation should only remove axioms which are relevant to the contracted axiom; this is guaranteed by restricting the union of the result of the contraction ($O - X$) and the contracted axiom (X) to entail (or be equivalent to) the original ontology (O):

(B-6) $O \subseteq Cn((O - X) \cup X)$

(O-5) comes very close to that by restricting $Cn((O - X) \cup X)$ to be maximal out of all the possible selections for $O - X$ that satisfy the other postulates: if there is any $Y \subseteq O$ giving a "larger" set $Cn(Y \cup X)$, then Y will necessarily entail X (so Y would not be a possible subset of $O - X$, by (O-3)). The latter implication ($\emptyset \not\models X$) was included in (O-5) in order to capture a certain limit case.

Now let us see why this set of postulates satisfies the required properties. The Existence property is easy to show. As a DL based ontology, O contains a finite number of axioms. Thus, there is only a finite number of subsets of O, so one can find at least one $Y \subseteq O$ for which $Cn(Y \cup X)$ is maximal. Once some technical details and limit cases are taken care of, the following proposition can be shown:

Proposition 2.3.1 *For any logic $\langle L, Cn\rangle$, there is a contraction operator '$-$' such that the operation $O - X$ satisfies (O-1)-(O-5) for all finite $O \subseteq L$ and all $X \subseteq L$.*

The second property, AGM Rationality, is more difficult to show, so we will break its proof in two parts. Firstly, we will show that if (B-1)-(B-6) are satisfied by a contraction

operator, then (O-1)-(O-5) are also satisfied. This is trivial for (O-1)-(O-4), as these are equivalent reformulations of (B-2)-(B-5) respectively. To show that (O-5) is satisfied as well, notice that (B-6) requires that O is entailed by $Cn((O - X) \cup X)$. If $O \models X$, then $O \models Cn(Y \cup X)$, for all $Y \subseteq O$. This fact, combined with the requirement imposed by (B-6), shows that the "if part" of (O-5) cannot be true for any $Y \subseteq O$, so (O-5) trivially holds. If, on the other hand, $O \not\models X$ then (B-3) (equivalently, (O-2)) indicates $O - X = O$; thus, again, the "if part" of (O-5) cannot be true for any $Y \subseteq O$, so (O-5) holds. This gives the following result:

Proposition 2.3.2 *If a contraction operator satisfies* (B-1)-(B-6)*, then it satisfies* (O-1)-(O-5).

This result implies that the original set of postulates (B-1)-(B-6) is stronger than (O-1)-(O-5); this should be expected, by Proposition 2.3.1, as the result of this proposition does not hold for (B-1)-(B-6) (see Lemma 1 and [FPA04]).

To show AGM Rationality, we should also show that the two sets of postulates are actually equivalent whenever possible (i.e., in base-AGM-compliant DLs). The proof follows similar steps as the proof of Proposition 2.3.2. The only non-trivial task is to show that whenever (O-1)-(O-5) are satisfied, (B-6) is also satisfied. This is shown by the fact that, in base-AGM-compliant logics, there is always a $Y \subseteq O$ which does not entail X, such that $Cn(Y \cup X)$ entails O; thus, (O-5) guarantees that $O - X$ will be selected in such a way that $Cn((O - X) \cup X)$ will imply O, thus satisfying (B-6). Once some limit cases are taken care of (one of which justifies the use of the implication $\emptyset \not\models X$ in (O-5)), the following can be shown:

Proposition 2.3.3 *For a base-AGM-compliant logic, if a contraction operator satisfies* (O-1)-(O-5)*, then it satisfies* (B-1)-(B-6).

Finally, it is important to note that the proposed postulates, as well as Propositions 2.3.1-2.3.3, are applicable not only to DLs, but also to all logics that comply with the $\langle L, Cn\rangle$ model.

2.3.2 Postulates for Revision

To the best of our knowledge, there has been no attempt to recast the AGM postulates for revision in the context of the foundational model; furthermore, there has been no attempt to generalize these postulates in the sense of [FPA04]. The main reason for the latter shortcoming are postulates which require the definition of a negation. The definitions of negation presented in the previous section allow us to overcome this problem and present some initial thoughts on these issues for DLs.

The original AGM postulates for revision (K+1)-(K+6) can be found in [AGM85]. Postulate (K+1) requires that the result of revision is a theory; in our context, this should

be dropped, as we are working on the foundational model. Postulates (K+2)-(K+5) can be reformulated as follows:

(O+1) $X \subseteq O + X$.

(O+2) If $Cn(O \cup X) \neq L$, then $O + X = O \cup X$.

(O+3) If $Cn(X) \neq L$, then $Cn(O + X) \neq L$.

(O+4) If $X \cong Y$, then $O + X \cong O + Y$.

It can be easily shown that each of (O+1)-(O+4) is equivalent to (K+2)-(K+5) in the standard case. Postulate (K+6) poses some extra problems, because it requires the definition of negations of DL axioms. A straightforward (and equivalent) reformulation of (K+6) follows:

(O+5) $(O + X) \cap O \cong O - \neg X$.

In (O+5), the '$\neg$' symbol may be replaced by the standard negation, consistency negation or coherence negation, depending on our needs and on which type(s) of negation exist in the underlying logic.

In the AGM theory, there is a close connection between revision and contraction, as this is expressed by the *Harper Identity* (which is equivalent to (O+5)) and *Levi Identity*; here we present a generalized version of these identities:

Harper: $O - X \cong Cn(O + \neg X) \cap Cn(O)$.
Levi: $O + X \cong Cn(O - \neg X) \cup Cn(X)$.

Again, in place of the symbol '$\neg$', any of the negations that we proposed could be used. In the AGM setting, it has been shown that for any given revision operator that satisfies the AGM postulates for revision, the contraction operator defined by the Harper identity satisfies the AGM postulates for contraction; moreover, for any given contraction operator that satisfies the AGM postulates for contraction, the revision operator defined by the Levi identity satisfies the AGM postulates for revision. One of our most important goals for future work is the proof that these facts hold for the generalized versions of the postulates, the Levi and the Harper identities.

If the coherence negation is used for (O+5), the Levi and the Harper identities, then it is more appropriate to replace the postulates (O+2) and (O+3) with following coherence-based postulates:

(O+2*) If $O \cup X$ is coherent, then $O + X = O \cup X$.

(O+3*) If X is coherent, then $O + X$ is coherent.

The coherence postulates are useful for the revision on the ontologies which have only T-boxes, because their incoherence appears much more often than their inconsistency. It is more meaningful to avoid their hidden inconsistency, i.e. their incoherence.

2.4 Importance for Dynamics in the Ontology Lifecycle

As has been mentioned in the introduction of this chapter, inconsistencies, as well as negations in ontologies are closely related to ontology change. In this chapter we have proposed a general framework accounting for negation, inconsistency and change for DL-based ontologies, which aims at providing a foundation for reasoning and management of dynamic ontologies. Such a foundation is of utmost importance for the deployment of real-world applications in the context of the Semantic Web.

In our framework, we have shown how to use the proposed negations to achieve the Harper identity and Levi identity for ontology change, by which we can make a close connection between the ontology revision and contraction operations. The distinction between incoherence and inconsistency provides us two different approaches for devising rationality postulates for ontology revision, which cover different needs in different application scenarios.

The well-founded postulates for ontology change (transformed according to the needs of the particular application) are of general importance for good design practices when implementing a dynamic ontology lifecycle, e.g. following the methodology introduced in [NHL$^+$06]. The postulates can be specifically applied (1) to the process of dynamic ontology development (quality assesment of proposed changes with respect to the coherence/consistency), or (2) to the integration of learned knowledge (when selecting the most appropriate parts to incorporate into the master precise collaborative ontology; we elaborate this topic in [NLHZ07]).

Chapter 3

Semantics of Ontology Diffs

by ALESSANDRO ARTALE, NORMAN FOO, ENRICO FRANCONI AND TOMMIE MEYER[1]

We consider in this chapter an abstract notion of semantic diff for arbitrary logic-based ontology or knowledge representation languages. The scenario is the one where an ontology engineer generates different version of an ontology while those different versions need to be stored and retrieved. We investigate the notion of *semantic diff* as a way to both enlighten the differences between two different versions (hereafter called the source and the target ontology), and as the minimal piece of information that we need to store together with the source ontology to obtain the target ontology.

The work presented in this chapter is still preliminary, and it should be considered primarily as a starting point motivating further deeper research on the topic, eventually meant to support well-founded development of practical applications utilising an ontology diff implementation.

3.1 Semantic Diff

We introduce now the *ideal* definition of semantic diff, that we call *strong*, as being the definition that embodies all the characteristics which are invariant under entailment and logical equivalence.

Definition 1 (Strong semantic diff)

Let $\mathscr{L}$ be a monotonic logic language. Given Σ, $\mathbb{O}_S$, $\mathbb{O}_T$, $\mathbb{A}$, $\mathbb{R}$ as sets of formulas in $\mathscr{L}$— being Σ a domain theory, $\mathbb{O}_S$ a source ontology, and $\mathbb{O}_T$ a target ontology — then $\langle \mathbb{A}, \mathbb{R} \rangle$ is an ontology diff, *and $\mathbb{A}$ is called its "*add*" component and $\mathbb{R}$ its "*remove*" component, if the following holds:*

[1] With minor modifications by Vít Nováček.

1. *(target from source with diff)*
   $(\mathbb{O}_S^i \cup \mathbb{A}) \setminus \mathbb{R} \models\!\!\models_{KB} \mathbb{O}_T^j$
2. *(duality)*
   $(\mathbb{O}_T^j \cup \mathbb{R}) \setminus \mathbb{A} \models\!\!\models_{KB} \mathbb{O}_S^i$
3. *(non-redundancy of diff)*
   $\mathbb{O}_S \models_{KB} \mathbb{R}$
   $\mathbb{O}_T \models_{KB} \mathbb{A}$
4. *(minimality)*
   $\mathbb{A}$ *and* $\mathbb{R}$ *are minimal w.r.t. set inclusion*
5. *(invariance under equivalent source and target)*
   $\mathbb{O}_S^i \models\!\!\models_{KB} \mathbb{O}_S$
   $\mathbb{O}_T^j \models\!\!\models_{KB} \mathbb{O}_T$

A strong semantic diff is therefore a pair of formulas to be added and formulas to be removed from a source ontology in order to get a target ontology (item 1). Moreover, by adding the "remove" formulas from the target and removing the "add" formula, the source ontology will be obtained (item 2); this is the dual use of the diff. Note that the introduction of integer i, j indices as superscripts of the source and target ontologies indicates invariance of the respective statements across possibly multiple equivalent ontologies.

The "add" and the "remove" components of a strong semantic diff are non redundant, in the sense that only formulas that are entailed by the source ontology can be removed, and only formulas that are entailed by the target ontology can be added; it is therefore impossible to specify a semantic diff where the "add" component contains formulas which are not entailed by the target ontology, and likewise it is impossible to specify a semantic diff where the "remove" component contains formulas which are not entailed by the source ontology (item 3).

A strong semantic diff is always represented in a minimal way (item 4). The strong characterisation of this kind of semantic diff is that the same diff applied to equivalent source ontologies should give equivalent target ontologies (item 5), thus sporting a deep sense of semantics in its definition.

As the reader can note, the operation of *adding* and *removing* is based in this definition on *set union* and *set difference*. This has interesting consequences.

Lemma 2 (Interesting facts)

a) (disjointness)
$\mathbb{A} \cap \mathbb{R} = \varnothing$

b) (invariance of order of application)

$(\mathbb{O}_S^i \cup \mathbb{A}) \setminus \mathbb{R} \models\!\models_{KB} (\mathbb{O}_S^i \setminus \mathbb{R}) \cup \mathbb{A}$

$(\mathbb{O}_T^j \cup \mathbb{R}) \setminus \mathbb{A} \models\!\models_{KB} (\mathbb{O}_T^j \setminus \mathbb{A}) \cup \mathbb{R}$

c) (belief revision operator)

The binary operators "$\cup$" and "$\setminus$", if considered as belief revision *operators "$\oplus$" and "$\ominus$", satisfy the postulates for* update *and* erasure *respectively of Katsuno-Mendelzon, presented in [KM91].*

Due to minimality, it is easy to see that a strong semantic diff has always disjoint "add" and "remove" components. Moreover, it is always possible to swap the set union and the set difference operators without changing the semantics of the strong semantic diff.

Example 2 (Transitive graphs)

Given Σ, $\mathbb{O}_S$, $\mathbb{O}_T$ and $\mathbb{O}_T^1$:

Σ *defines transitivity over directed graphs*

$\mathbb{O}_S = \{b \rightarrow c\}$

$\mathbb{O}_T = \{a \rightarrow b, b \rightarrow c, a \rightarrow c\}$

$\mathbb{O}_T^1 = \{a \rightarrow b, b \rightarrow c\} \models\!\models_{KB} \mathbb{O}_T$

there is a unique strong semantic diff:

$\mathbb{A} = \{a \rightarrow b\}$

$\mathbb{R} = \{\}$

Non minimal semantic diffs are:

$\mathbb{A}^1 = \{a \rightarrow b, a \rightarrow c\}$

$\mathbb{A}^2 = \{a \rightarrow b, b \rightarrow c\}$,

$\mathbb{A}^3 = \{a \rightarrow b, b \rightarrow c, a \rightarrow c\}$

with $\mathbb{A}^2 \models\!\models_{KB} \mathbb{A}^3$

Example 3 (Propositional logic)

Given Σ, $\mathbb{O}_S$, and $\mathbb{O}_T$:

$\Sigma = \{a \leftrightarrow b\}$

$\mathbb{O}_S = \{\}$

$\mathbb{O}_T = \{a, b, c\}$

there are two alternative strong semantic diffs:

$\mathbb{A}^1 = \{a, c\}, \quad \mathbb{A}^2 = \{b, c\}$

$\mathbb{R} = \{\}$

By dropping the minimality requirement, there is a unique semantic diff:

$\mathbb{A} = \{a, b, c\}$

Note that $\mathbb{A} \models_{KB} \mathbb{A}^1 \models_{KB} \mathbb{A}^2$

Example 4 (Non-existence of strong semantic diff)

Given Σ, $\mathbb{O}_S$, $\mathbb{O}_S^1$, *and* $\mathbb{O}_T$*:*

Σ *defines transitivity over directed graphs*

$\mathbb{O}_S = \{a \rightarrow b, b \rightarrow c\}$

$\mathbb{O}_S^1 = \{a \rightarrow b, b \rightarrow c, a \rightarrow c\} \models_{KB} \mathbb{O}_S$

$\mathbb{O}_T = \{b \rightarrow c, a \rightarrow c\}$

there is no strong semantic diff.
There is a unique semantic diff for each equivalent source ontology if the invariance under equivalence condition is dropped:

$\mathbb{R} = \{a \rightarrow b\}$

$\mathbb{A} = \{a \rightarrow c\}$

$\mathbb{A}^1 = \{\}$

Alternatively, there is a unique semantic diff if the non-redundancy of diff condition is dropped:

$\mathbb{R} = \{a \rightarrow b\}$

$\mathbb{A} = \{a \rightarrow c\}$

The examples above emphasise immediately some problems: on the one hand we may not have a unique strong semantic diff, unless we drop the minimality condition; on the other hand, we may not have at all a strong semantic diff, unless we drop the invariance under equivalent sources and targets condition or the non-redundancy of diff condition.

This suggests that we need to study weaker definitions of semantic diff. In the following, we will show how different potential methods to compute a diff between ontologies may lead to different semantic and computational properties.

3.2 Semantic Diff Based on Normal Forms

Given a set of formulas $\mathbb{O}$, we call $\widehat{\mathbb{O}}$ its deductive closure under the theory Σ. The deductive closure of an ontology may be considered as a (unique) representative for the class of all the logically equivalent ontologies. By considering the deductive closure, we may hope that the definition of the semantic diff as the simple set difference between the deductive

closure of the source and target ontologies (which is also quite easy computationally) is indeed a strong semantic diff. However, this is not the case.

Theorem 3.2.1 (Inadequacy of deductive closure)
If we define $\mathbb{A}$ *and* $\mathbb{R}$ *as follows:*

$\mathbb{A} \doteq \widehat{\mathbb{O}_T} \setminus \widehat{\mathbb{O}_S}$

$\mathbb{R} \doteq \widehat{\mathbb{O}_S} \setminus \widehat{\mathbb{O}_T}$

then $\mathbb{A}$ *and* $\mathbb{R}$ *may violate both the minimality condition and the invariance under equivalent sources and targets condition.*

PROOF: Consider the following counterexamples.

- Given Σ, $\mathbb{O}_S$, $\mathbb{O}_S^1$, and $\mathbb{O}_T$:

  Σ defines transitivity over directed graphs

  $\mathbb{O}_S = \{a \rightarrow b, b \rightarrow c\}$

  $\mathbb{O}_S^1 = \widehat{\mathbb{O}_S} = \{a \rightarrow b, b \rightarrow c, a \rightarrow c\} \models\!\!\models_{KB} \mathbb{O}_S$

  $\mathbb{O}_T = \{b \rightarrow c, a \rightarrow c\}$

  then:

  $\mathbb{A} = \widehat{\mathbb{O}_T} \setminus \widehat{\mathbb{O}_S} = \{\}$

  $\mathbb{R} = \widehat{\mathbb{O}_S} \setminus \widehat{\mathbb{O}_T} = \{a \rightarrow b\}$

  Observe that this violates the invariance under equivalent sources and targets condition:

  $(\widehat{\mathbb{O}_S} \cup \mathbb{A}) \setminus \mathbb{R} \models\!\!\models_{KB} \mathbb{O}_T$

  $(\mathbb{O}_S \cup \mathbb{A}) \setminus \mathbb{R} \not\models\!\!\models_{KB} \mathbb{O}_T$

- Given Σ, $\mathbb{O}_S$, and $\mathbb{O}_T$:

  Σ defines transitivity over directed graphs

  $\mathbb{O}_S = \{b \rightarrow c\}$

  $\mathbb{O}_T = \{a \rightarrow b, b \rightarrow c, a \rightarrow c\}$

  then:

  $\mathbb{A} = \widehat{\mathbb{O}_T} \setminus \widehat{\mathbb{O}_S} = \{a \rightarrow b, a \rightarrow c\}$

  $\mathbb{R} = \widehat{\mathbb{O}_S} \setminus \widehat{\mathbb{O}_T} = \{\}$

  However, the minimal semantic diff is:

  $\mathbb{A} = \{a \rightarrow b\}$

  $\mathbb{R} = \{\}$

□

Another possibility would be to consider the normal form obtained with the *lean* representation of an ontology, i.e., with a form of minimal representation.

Definition 2 (Lean ontology)

An ontology $\mathbb{O}$ *is* lean *if no subset* $\mathbb{P} \subseteq \mathbb{O}$ *is entailed by the ontology, i.e.,* $\mathbb{O} \models \mathbb{P}$.

The case of lean ontologies has not been studied yet.

3.3 Semantic Diff for RDFS

In the following we consider RDFS as the ontology language.

The main idea is to start from a simple *syntactic diff* between the two versions and then minimising the result of the syntactic diff by exploiting the semantics of the RDFS ontology language in order to get a semantic diff. The syntactic diff is simply the set difference between two sets of RDFS triples as defined below.

Definition 3 (Syntactic Diff)

Let $\mathbb{O}_S$ *and* $\mathbb{O}_T$ *be the source and the target RDFS ontologies of a versioning process. The* syntactic diff *is composed by the pair:*

$\mathbb{A}_{syn} = \mathbb{O}_T \setminus \mathbb{O}_S$

$\mathbb{R}_{syn} = \mathbb{O}_S \setminus \mathbb{O}_T$

To define the semantic diff we take into account the semantic of RDFS and in particular the notion of *entailment* between two RDFS graphs as defined in the W3C standard. An RDFS graph S entails an RDFS graph T ($S \models_{\text{RDF}} T$) if there is a total homomorphism from T to $\widehat{S}$, the deductive closure or the *completed form* of S, i.e. axiomatic triples are added to S while entailment rules introduce implicit triples in S.

Another relevant notion used in the following is that one of *equivalence* between two RDFS graphs. We say that an RDFS graph S is equivalent to an RDFS graph T ($S \models\!\models_{\text{RDF}} T$) if they entail one each other. Obviously, the completed form of an RDFS graph is equivalent to the original graph: $\widehat{S} \models\!\models_{\text{RDF}} S$. Finally, given a graph S and a homomorphism $\mathcal{H}$, with $S_{\mathcal{H}}$ we denote the graph S with bnodes renamed by $\mathcal{H}$.

Definition 4 (RDF Semantic Diff)

Let $\mathbb{O}_S$ *and* $\mathbb{O}_T$ *be the source and the target RDFS ontologies of a versioning process. A* semantic diff *is a pair* $\mathbb{A}$, $\mathbb{R}$ *such that:*

1. $\mathbb{R}$ *is the maximal subset of* $\mathbb{R}_{syn}$ *such that none of the subsets of* $\mathbb{R}$ *is entailed by* $\mathbb{O}_T$ *(i.e., no elements entailed by the target ontology are part of the remove, being that semantically useless):*

   (a) $\mathbb{R} \subseteq \mathbb{R}_{syn}$

   (b) *for each* $\mathbb{P} \subseteq \mathbb{R}$ *then* $\mathbb{O}_T \not\models_{RDF} \mathbb{P}$

   (c) $\mathbb{R}$ *is maximal wrt set inclusion*

2. $\mathbb{A}$ *is the maximal subset of* $\mathbb{A}_{syn}$*, modulo an arbitrary bnode renaming* α*, such that none of the subsets of* $\mathbb{A}$ *is entailed by* $\mathbb{O}_S$ *(i.e., no elements entailed by the source ontology are part of the add, being that semantically useless) and moreover a merging of* $\mathbb{A}$ *with the source ontology is not redundant:*

   (a) $\mathbb{A} \subseteq \alpha(\mathbb{A}_{syn})$

   (b) *for each* $\mathbb{Q} \subseteq \mathbb{A}$ *then* $\mathbb{O}_S \not\models_{RDF} \mathbb{Q}$

   (c) $(\mathbb{O}_S \cup \mathbb{A}) \setminus \mathbb{R} \models\!\!\!|_{RDF} (\mathbb{O}_S \cup \mathbb{A}_{syn}) \setminus \mathbb{R}$

   (d) $\mathbb{A}$ *is maximal wrt set inclusion*

We call weak RDF semantic diff *the RDF semantic diff without the condition (2.c) above.*

We now show the most relevant result, i.e. that given an RDF semantic diff as defined above, we can reconstruct the target ontology starting from the source ontology and the semantic diff. Moreover, in general the RDF semantic diff satisfies all the conditions for a strong semantic diff, but one.

Theorem 3.3.1 (RDF semantic diff)

The RDF semantic diff satisfies the conditions in definition 1 with the exception of the invariance under equivalent sources and targets condition. The weak RDF semantic diff satisfies the conditions in definition 1 with the exception of the invariance under equivalent sources and targets condition and of the minimality condition.

Example 5 (RDF semantic diff)

Given $\mathbb{O}_S$*, and* $\mathbb{O}_T$ *(note that "* $\rightarrow$ *" corresponds to "*`rdfs:subclass`*", and that* X, Y *are bnodes):*

$\mathbb{O}_S = \{a \rightarrow b, b \rightarrow c, a \rightarrow c, c \rightarrow d, d \rightarrow X\}$

$\mathbb{O}_T = \{a \rightarrow b, b \rightarrow c, c \rightarrow d, b \rightarrow d, d \rightarrow Y\}$

then:

$\mathbb{A}_{syn} = \{b \rightarrow d, d \rightarrow Y\}$

$\mathbb{R}_{syn} = \{a \rightarrow c, d \rightarrow X\}$

However, the RDF semantic diff is:

$\mathbb{A} = \{\}$

$\mathbb{R} = \{\}$

In this example, the RDF semantic diff and the weak RDF semantic diff coincide.

Example 6 (RDF semantic diff)

Given $\mathbb{O}_S$, and $\mathbb{O}_T$:

$\mathbb{O}_S = \{a \rightarrow X, a \rightarrow Y, X \rightarrow b, Y \rightarrow c\}$

$\mathbb{O}_T = \{a \rightarrow Z, Z \rightarrow b, Z \rightarrow c\}$

then:

$\mathbb{A}_{syn} = \{a \rightarrow Z, Z \rightarrow b, Z \rightarrow c\}$

$\mathbb{R}_{syn} = \{a \rightarrow X, a \rightarrow Y, X \rightarrow b, Y \rightarrow c\}$

There are two alternative RDF semantic diffs:

$\mathbb{A}^1 = \{X \rightarrow c\}$

$\mathbb{A}^2 = \{Y \rightarrow b\}$

$\mathbb{R} = \{\}$

3.4 Diff Formalisation in the Scope of Ontology Evolution

This chapter presents a rather conceptually different, though still inherently related view on the topics studied in Chapter 2. While Chapter 2 investigates notions of negation, (in)consistency and ontology change postulates and studies their formal properties, the material presented here provides a kind of reverse view. It defines the (semantic) ontology diff and its structure, consisting from addition and revision sets of formulae of an ontology language. Initial study on its properties and related practical features w.r.t. widely used RDFS ontology language is given.

Even though the study is very preliminary in its present state, the suggested diff formalisation and several important issues that have already been identified (e.g., the more practical – weaker – diff alternatives) are still very important for practical applications of ontology maintenance. These make use of an ontology diff implementation quite often [NM04, KFKO02, VG06] without a proper and/or common underlying formalisation. However, the conformance to a common formalisation can make ontology maintenance and handling of the logical consequences of a diff (i.e., change) much more transparent and universal even across different application scenarios or (logics-based) ontology languages. Moreover, when combined with conformance to the postulates for ontology change described in Section 2.3, the formalisation of diffs can further facilitate efficient and well-founded process of consistent and rational ontology maintenance with a practical diff processing involved.

Chapter 4

Multi-version Ontology Reasoning Using Temporal Logics

by ZHISHENG HUANG AND HEINER STUCKENSCHMIDT[1]

As has been widely agreed (see for instance [Sto04, KFKO02, NM04]), versioning is the key to compatibility as it enables each application to use a version of the ontology that best fits its requirements. This chapter focuses on supporting the **management of different versions of the same ontology on a semantic level**. In particular, we want to provide functionality for answering queries about knowledge derivable from different versions. The corresponding approach has to provide two kinds of functionalities:

- Ask questions about statements derivable from a certain version.
- Ask for a version that allows to derive certain statements

While the first kind of functionality can be used to inspect a given version of the ontology in order to find out whether important statements can or cannot be derived from it, the second kinds of functionality helps to find a version that is compatible with a given application because important statements can be derived from it.

Another issue is the scope of the approach, in particular the space of versions to be considered. There are several possible scenarios. In most relevant cases, we are concerned with a history of different versions of the same ontology where each version replaces the previous one. We call this the **retrospective** approach. As a result, we have a sequence of versions. There are also scenarios, in which different versions of the ontology co-evolve. This is mostly the case in scenarios where the development of the ontology is not controlled by an authority. In the following, we ignore this scenario which is less relevant for professional ontology development.

In this document, we make a contribution towards a general apparatus for supporting multi-version management on the semantic level. We will do this by defining and implementing a query language that is able to answer relevant questions about multiple versions

[1] Based on [HS05b], with minor modifications by Vít Nováček.

of the same ontology as identified in the usage scenarios. Rather than trying to directly support all of these queries, be identify a basic machinery that provides the expressive power needed to provide the required functionality, but still needs to be optimized for specific queries. We base this machinery on a temporal logic over statements derivable from different versions of the ontology. Queries concerning the content derivable from versions can be stated in this temporal logic and are evaluated using model-checking techniques. In this work we focus on the retrospective approach to multi-version analysis and only sketch a possible extension to the prospective analysis. Concerning the representation of the ontologies being analyzed we restrict ourselves to ontologies encoded in OWL-DL. With respect to derivable statements, we mainly consider subsumption between named classes. We want to stress, however, that the approach presented is independent from the representation of the ontologies and can easily be adapted to other representation languages.

In the following we first introduce the temporal logic approach to managing multiple versions. Based on this, we define a minimal query language for multiple versions. Moreover, we briefly describe the basic principles of the consequent reasoning, utilising the introduced formalism. The approach presented here is discussed in a broader context of this deliverable in Chapter 6, mostly in Section 6.1.2.

4.1 A Temporal Logic for Multi-version Ontology Reasoning

Temporal logics can be classified as two main classes with respect to two different time models: linear time model and branching time model. The linear time logics express properties over a single sequence of states. This view is suitable for the retrospective approach to multi-ontology reasoning where we assume the existence of a sequence of versions. Branching time logics are express properties across different sequences of states. This feature would be needed for the prospective approach where we consider different possible sequences of changes in the future. The linear temporal logic **LTL** is a typical temporal logic for modeling linear time, whereas the computation tree logic **CTL** is a typical one for modeling branching time [RU71, vB95, CGP99].

Temporal logics are often future-oriented, because their operators are designed to be ones which involve the future states. Typical operators are: the operator $\mathbf{Future}\,\phi$ which states that 'ϕ holds sometimes in the future with respect to the current state', and the operator $\mathbf{Alwaysf}\phi$ which states that 'ϕ always holds in the future with respect to the current state', and the operator $\phi\,\mathbf{Until}\,\psi$ which states that 'ϕ always holds in the future until ψ holds'. For a discrete time model, the operator $\mathbf{Next}\,\phi$ is introduced to state that ϕ holds at the next state with respect to the current state. For the retrospective reasoning, we only need a temporal logic that only talks about the past. Namely, it is one which can be used to compare the current state with some previous states in the past. It is natural to design the following past-oriented operators, which correspond with the counterparts of

the future oriented temporal operators respectively:

- the previous operator states that a fact ϕ holds just one state before the current state.
- the the sometimes-in-the past operator states that a fact ϕ holds sometimes in the past with respect to the current state.
- the always-in-the-past operator states that ϕ holds always in the past with respect to the current state.

In this document, we use a linear temporal logic, denoted as **LTLm**, which actually is a restricted linear temporal logic **LTL** to past-oriented temporal operators.

4.1.1 Version Spaces and Temporal Models

In the following, we will define the formal semantics for the temporal operators by introducing an entailment relation between a semantic model (i.e., multi-version ontologies) and a temporal formula. We consider a version of an ontology to be a state in the semantic model. We do not restrict ontology specifications to a particular language (although OWL and its description logics are the languages we have in mind). In general, an ontology language can be considered to be a set of formulas that is generated by a set of syntactic rules in a logical language $\mathcal{L}$.

We consider multi-versions of an ontology as a sequence of ontologies which are connected with each other via change operations. Each of these ontologies has a unique name. This is different from the work by [HP04], who consider that an ontology is one which contains the set of other ontologies which are backwards compatible with it. We have the following definition.

Definition 5 (Version Space) *A version space S over an ontology set Os is a set of ontology pairs, namely, $S \subseteq Os \times Os$.*

We use version spaces as a semantic model for our temporal logic, restricting our investigation to version spaces that present a linear sequence of ontologies:

Definition 6 (Linear Version Space) *A linear version space S on an ontology set Os is a version space which is a finite sequence of ontologies*

$$S = \{\langle O_1, O_2\rangle, \langle O_2, O_3\rangle, \cdots, \langle O_{n-1}, O_n\rangle\}$$

Alternatively we write the sequence S as follows:

$$S = (O_1, O_2, \cdots, O_n)$$

We use $S(i)$ to refer the i_th ontology O_i in the space. For a version space $S = (O_1, O_2, \cdots, O_n)$, We call the first ontology $S(1)$ in the space the *initial version of the version space,* and the last ontology $S(n)$ the *latest version of the version space* respectively.

We introduce an ordering $\prec_S$ with respect to a version space S as follows:

Definition 7 (Ordering on Version Space) *$O \prec_S O'$ iff O occurs prior to O' in the sequence S, i.e., $S = (\cdots, O, \cdots, O', \cdots)$.*

Proposition 4.1.1 (Prior version and Linear Ordering)
the prior version relation $\prec_S$ is a linear ordering, namely, $\prec_S$ is
(i) irreflexive, i.e., $(O \not\prec_S O)$,
(ii) transitive, i.e., $O \prec_S O'$ and $O' \prec_S O'' \Rightarrow O \prec_S O''$,
(iii) asymmetry, i.e., $O \prec_S O' \Rightarrow O' \not\prec_S O$,
(iv) comparable, i.e., either $O \prec_S O'$ or $O' \prec_S O$,
for any ontology O, O', O''.

4.1.2 Syntax and Semantics of LTLm

The Language $\mathcal{L}+$ for the temporal logic $\mathbf{LTLm}$ can be defined as an extension to the ontology language $\mathcal{L}$ with Boolean operators and the temporal operators as follows:

$q \in \mathcal{L} \Rightarrow q \in \mathcal{L}+$
$\phi \in \mathcal{L}+ \Rightarrow \neg\phi \in \mathcal{L}+$
$\phi, \psi \in \mathcal{L}+ \Rightarrow \phi \wedge \psi \in \mathcal{L}+$
$\phi \in \mathcal{L}+ \Rightarrow \mathbf{PreviousVersion}\,\phi \in \mathcal{L}+$
$\phi \in \mathcal{L}+ \Rightarrow \mathbf{AllPriorVersions}\,\phi \in \mathcal{L}+$
$\phi, \psi \in \mathcal{L}+ \Rightarrow \phi\,\mathbf{Since}\,\psi \in \mathcal{L}+$

Where the negation $\neg$ and the conjunction $\wedge$ must be new symbols that do not appear in the language $\mathcal{L}$ to avoid the ambiguities (thus they are different also from the notions defined in Chapter 2, most specifically from negation given in the scope of DL ontologies). Define the disjunction $\vee$, the implication $\rightarrow$, and the bi-conditional $\leftrightarrow$ in terms of the conjunction and the negation as usual. Define $\bot$ as a contradictory $\phi \wedge \neg\phi$ and $\top$ as a tautology $\phi \vee \neg\phi$ respectively.

Using these basic operators, we can define some additional operators useful for reasoning about multiple versions. We define the $\mathbf{SomePriorVersion}$ operator in terms of the $\mathbf{AllPriorVersions}$ operator as

$$\mathbf{SomePriorVersion}\phi =_{df} \neg\mathbf{AllPriorVersions}\neg\phi$$

The always-in-the-past $\mathbf{AllPriorVersions}$ operator is one which does not consider the current state. We can define a strong always-in-the-past $\mathbf{AllVersions}$ operator as

$$\mathbf{AllVersions}\phi =_{df} \phi \wedge \mathbf{AllPriorVersions}\,\phi,$$

which states that 'ϕ always holds in the past including the current state'.

Let S be a version space on an ontology set Os, and o be an ontology in the set Os, we extend the entailment relation for the extended language $\mathcal{L}+$ as follows:

$S, O \models q$	iff	$O \models q, for\ q \in \mathcal{L}$.
$S, O \models \neg\phi$	iff	$S, O \not\models \phi$.
$S, O \models \phi \wedge \psi$	iff	$S, O \models \phi, \psi$.
$S, O \models \mathbf{PreviousVersion}\,\phi$	iff	$\langle O', O\rangle \in S$ such that $S, O' \models \phi$.
$S, O \models \mathbf{AllPriorVersions}\,\phi$	iff	for any O' such that $O' \prec_S O, S, O' \models \phi$.
$S, O \models \phi\mathbf{Since}\psi$	iff	$\exists(O_1 \ldots O_i)(O_1 \prec_S O_2 \ldots O_{i-1} \prec_S O_i = o)$ such that $S, O_j \models \phi$ for $1 \leq j \leq i$ and $S, O_1 \models \psi$).

For a linear version space S, we are in particular interested in the entailment relation with respect to its latest version of the ontology $S(n)$ in the version space S. We use $S \models \phi$ to denote that $S, S(n) \models \phi$. Model checking has been proved to be an efficient approach for the evaluation of temporal logic formulas[CGP99]. In the implementation of MORE, we are going to use the standard model checking algorithm for evaluation a query in the temporal logic **LTLm**. Therefore, we do not need a complete axiomatization for the logic **LTLm** in this document.

4.1.3 Formal Properties

The validity of a temporal formula in the logic $\mathbf{LTLm}$ is defined as a property which is independent of any particular **LTLm** model and any state in the model. Namely, the property is true in every state of any **LTLm** model. We have the following definition:

Definition 8 (Validity) $\models \phi$ *iff* $S, o \models \phi$ *for any* S, o.

Here is a list of formal properties in the logic $\mathbf{LTLm}$:

Proposition 4.1.2 (Formal Properties of Temporal Operators)
(a) $\models \mathbf{AllPriorVersions}\phi \rightarrow \mathbf{SomePriorVersion}\phi$.
(the always-in-the-past implies the sometimes-in-the-past.)

(b) $\models \mathbf{PreviousVersion}\phi \rightarrow \mathbf{SomePriorVersion}\phi$.
(the previous implies the sometimes-in-the-past.)

(c) $\models \mathbf{PreviousVersionSomePriorVersion}\phi \rightarrow \mathbf{SomePriorVersion}\phi$.
(the previous of the sometimes-in-the-past implies the sometimes-in-the-past.)

(d) $\models \mathbf{SomePriorVersionSomePriorVersion}\phi \rightarrow \mathbf{SomePriorVersion}\phi$.
(idempotent of the sometimes-in-the-past.)

(e) $\models \mathbf{AllPriorVersionsAllPriorVersions}\phi \wedge \mathbf{PreviousVersion}\phi \rightarrow \mathbf{AllPriorVersions}\phi$.
(quasi-idempotent of the always-in-the-past.)

(f) $\models \mathbf{PreviousVersionPreviousVersion}\phi \rightarrow \mathbf{SomePriorVersion}\phi$.
(the previous of the previous implies the sometimes-in-the-past.)

(g) $\models \phi \mathbf{Since} \psi \rightarrow \mathbf{SomePriorVersion}\psi \vee \psi$.
(relation bewteen the since operator and the sometimes-in-the-past.)

(h) $\models \phi \mathbf{Since} \psi \rightarrow \phi$.
(ϕ since ψ implies that ϕ holds in the current version.)

(i) $\models \phi \wedge \psi \rightarrow \phi \mathbf{Since} \psi$.
(trivial case for the since operator.)

4.2 LTLm as a Query Language

There are two types of queries: reasoning queries and retrieval queries. The former concerns with an answer either 'yes' or 'no', and the latter concerns an answer with a particular value, like a set of individuals which satisfy the query formula. Namely, the evaluation of a a reasoning query is a decision problem, whereas the evaluation of a retrieval query is a search problem. In this section, we are going to discuss how we can use the proposed temporal logic to support both reasoning queries and retrieval queries.

Reasoning queries

Using the **LTLm** logic we can formulate reasoning queries over a sequence of ontologies that correspond to the typical questions mentioned in the introduction of this chapter.

Are all facts still derivable? This question can be answered for individual facts using reasoning queries. In particular, we can use the query $\phi \wedge \mathbf{PreviousVersion}\,\phi$ to determine for facts ϕ derivable from the previous version whether they still hold in the current version. The same can be done for older versions by chaining the $\mathbf{PreviousVersion}$ operator or by using the operator $\mathbf{AllVersions}$ to ask whether formulas were always true in past versions and are still true in the current one ($\mathbf{AllVersions}\,\phi$).

What facts are not derivable any more? In a similar way, we can ask whether certain facts are not true in the new version any more. This is of particular use for making sure that unwanted consequences have been excluded in the new version. The corresponding query is $\neg\phi \wedge \mathbf{PreviousVersion}\ \phi$. Using the $\mathbf{AllPriorVersions}$ operator, we can also ask whether a fact that was always true in previous versions is not true anymore.

Are the facts are newly derivable from the new version? Reasoning queries can also be used to determine whether a fact is new in the current version. As this is true if it is not true in the previous version, we can use the following query for checking this $\phi \wedge \neg\mathbf{PreviousVersion}\ \phi$. We can also check whether a new fact never held in previous versions using the following query $\phi \wedge \neg\mathbf{SomePriorVersion}\ \phi$.

What is the last version that can be used to derive certain facts? Using reasoning queries we can check whether a fact holds in a particular version. As versions are arranged in a linear order, we can move to a particular version using the $\mathbf{PreviousVersion}$ operator. The query $\mathbf{PreviousVersion}\ \mathbf{PreviousVersion}\ \phi$ for instance checks whether ϕ was true in the version before the previous one. The query $\phi\mathbf{Since}\psi$ states that ϕ always holds since ψ holds in a prior version.

A drawback of reasoning queries lies in the fact, that they can only check a property for a certain specific fact. When managing a different versions of a large ontology, the user will often not be interested in a particular fact, but ask about changes in general. This specific functionality is provided by retrieval queries.

Retrieval Queries

Many Description Logic Reasoners support so-called retrieval queries that return a set of concept names that satisfy a certain condition. For example, a children concept c' of a concept c, written $child(c, c')$, is defined as one which is subsumed by the concept c, and there exists no other concepts between them. Namely,

$$child(c, c') =_{df} c' \sqsubseteq c \wedge\ \nexists c''(c' \sqsubseteq c'' \wedge c'' \sqsubseteq c \wedge c'' \neq c \wedge c'' \neq c').$$

Thus, the set of new/obsolete/invariant children concepts of a concept on an ontology o in the version space S is defined as follows:

$$new_{children}(S, o, c) =_{df} \{c' | S, o \models child(c, c') \wedge \neg\mathbf{PreviousVersion}\, child(c, c')\}.$$

$$obsolete_{children}(S, o, c) =_{df} \{c' | S, o \models \neg child(c, c') \wedge \mathbf{PreviousVersion}\, child(c, c')\}.$$

$$invariant_{children}(S, o, c) =_{df} \{c' | S, o \models child(c, c') \wedge \mathbf{PreviousVersion}\, child(c, c')\}.$$

The same definitions can be extended into the cases like parent concepts, ancestor concepts, descendant concept and equivalent concepts. Those query supports are sufficient to evaluate the consequences of the ontology changes and the differences among multi-version ontologies.

4.2.1 Making version-numbers explicit

Temporal logics allow us to talk about the temporal aspects without reference to a particular time point. For reasoning with multi-version ontologies, we can also talk about the temporal aspects without mentioning a particular version name. We know that each state in the temporal logic actually corresponds with a version of the ontology. It is not difficult to translate the temporal statements into a statement which refers to an explicit version number. Here are two approaches for it: relative version numbering and absolute version numbering.

Relative version numbering

The proposed temporal logic is designed to be one for past-oriented. Therefore, it is quite natural to design a version numbering which is relative to the current ontology in the version space. We use the formula 0ϕ to denote that the property holds in the current version. Namely, we refer to the current version as the version 0 in the version space, and other states are used to refer to a version relative to the current version, written as $-i$ as follows:

$$0\phi =_{df} \phi.$$

$$(-i)\phi =_{df} \mathbf{PreviousVersion}((1-i)\phi).$$

The formula $-i\phi$ can be read as "the property ϕ holds in the previous i-th version".

Absolute version numbering

Given a version space S with n ontologies on it, i.e., $|S| = n - 1$. For the latest version $o = S(n)$, it is well reasonable to call the i-th ontology $S(i)$ in the version space the version i of S, denoted as i, S. Namely, we can use the formula $i, S\phi$ to denote that the property ϕ holds in the version i in the version space S. Thus, we can define the absolute version statement in terms of a relative version statement as follows:

$$(i, S)\phi =_{df} (i-n)\phi.$$

Explicit version numbering provides the basis for more concrete retrieval queries. In particular, we now have the opportunity to compare the children of a concept c in two

specific ontologies i and j in the version space S. The corresponding definitions are the following:

$$newChildren(S,c)_{i,j} =_{df} \{c' | S \models (i,S)\, child(c,c') \wedge \neg(j,S)\, child(c,c')\}.$$

$$obsoleteChildren(S,c)_{i,j} =_{df} \{c' | S \models \neg(i,S)\, child(c,c') \wedge (j,S)\, child(c,c')\}.$$

$$invariantChildren(S,c)_{i,j} =_{df} \{c' | S \models (i,S)\, child(c,c') \wedge (j,S)\, child(c,c')\}.$$

Again, the same can be done for other predicates like parent-, ancestor or descendant concepts.

4.3 Basic Principles of the Inference Implementation

There is a prototype implementation based on the approach described above. The system is implemented as an intelligent interface between an application and state-of-the art description logic reasoners (which support the DIG interface [BMC03]). The prototype provides server-side functionality in terms of an XML-based interface for uploading different versions of an ontology and posing queries to these versions. Requests to the server are analyzed by the main control component that also transforms queries into the underlying temporal logic queries if necessary. The main control element also interacts with the ontology repository and ensures that the reasoning components are provided with the necessary information and coordinates the information flow between the reasoning components.

The actual reasoning is done by model checking components for testing temporal logic formulas that uses the results of an external description logic reasoner for answering queries about derivable facts in a certain version. More on the reference implementation, some experiments and their evaluation can be found in [HS05a, HSvHK06].

Chapter 5

C-OWL Potential for Reasoning with Versioned Ontologies

by VÍT NOVÁČEK, MATHIEU D'AQUIN, JEAN LIEBER AND AMEDEO NAPOLI

In this chapter we present an initial proposal of an alternative approach to multi-version ontology reasoning, based on re-casting a solution for contextualised knowledge representation and reasoning into the ontology dynamics domain.

Context and its representation has been relatively intensively studied by the researchers in AI during the last two decades [McC93, Akm02]. Quite naturally, the importance of context has been recently recognised even within the Semantic Web and related knowledge representation paradigms [GG01, BGvH$^+$03].

Contextual knowledge representation aims at development of efficient formalisms and reasoning principles that would allow for dealing with sets of constructs (axioms, facts, rules, etc.) restricted to a particular domain (i.e., context), while not being necessarily globally consistent or meaningful. Another important feature of the context-dependent knowledge representation is relating the particular contexts among themselves (e.g. by the so called lifting mechanism [McC93] or by bridge rules [BDSZ02, BGvH$^+$03]).

Considering an evolving ontology, we can very naturally see it as a sequence of contexts changing in time (elaborated in Section 5.1). When providing mappings between the evolving versions of particular concepts, we can directly utilise the semantics and reasoning services originally meant to support contextual ontologies (covered by Section 5.2.1).

More specifically, we build on the similarities of notions of version and context spaces (of ontologies). We do so primarily in the scope of the C-OWL approach proposed in [BGvH$^+$03], since it presents an extension of a standard and widely used ontology language – OWL [BvHH$^+$04]. Note that rather than giving an exhaustive description of the original C-OWL features here, we reference to the paper [BGvH$^+$03] for the relevant detailed analysis, syntax and semantics of the extension and all other issues that are not directly related to the application of C-OWL to the multi-version ontology reasoning.

5.1 Version and Context Spaces – Essential Similarities

There is no explicit definition of context in [BGvH+03]. However, it is implicitly assumed that a context is an ontology, interpreted locally (basically, building on the local model semantics [GG01]). The local interpretation is based on appropriate modification of the standard OWL semantics [PSHH04]. Here we describe, how a set of particular ontology versions can be treated as a set of local contexts (Section 5.1.1). The mechanism of mapping between particular ontology versions is presented in Section 5.1.2 then.

5.1.1 Versions as Contexts

In the following, we use the notation for (a sequence of) versioned ontologies as introduced in Section 4.1.1 of Chapter 4. We employ the concept of linear version space $S = (O_1, O_2, \ldots, O_n)$, with ontology versions referenced by $S(i),\ i \in \{1, \ldots, n\}$, and with the linear ordering $\prec_S$ defined on it. This notation is convenient for expressing the actual precedence of ontology versions from the initial to the current one (not taking version branches into account, since the sequence is linear by definition).

The version space can be easily embedded into the scope of essential notions of contextual OWL extension. Without loss of generality, the *version space* can be treated as the *OWL space* (defined in [BGvH+03]). We only have to assume that:

1. ontology version indices $\{1, \ldots, n\}$ correspond to the respective ontology version URIs
2. the ontologies $O_1, \ldots, O_n$ are expressed using OWL (DL)
3. if an ontology O_i references a concept (role, individual) defined in an ontology O_j, it is explicitly annotated by the index j and considered as an element of *foreign* language w.r.t. O_i; moreover, we can safely restrict such foreign references by the $j < i$ inequality, since an ontology version intuitively would not reference future concepts (roles, individuals)

Now we can apply the specific semantics for contextual ontologies, as defined and elaborated in [BGvH+03], also to version spaces. The next section shows certain practicality of this approach when it comes to representation of explicit relations between ontology versions and reasoning across their sequences.

5.1.2 C-OWL Mapping Constructs as Inter-Version Relations

We have adopted the semantics for contextual ontologies as such in the previous section. However, we also need an expressive mechanism in order to relate the versions by explicit mappings. Only specifying the (previous) foreign definitions of ontology elements

(i.e., concepts, roles or individuals) in an ontology version may not be enough for many practical applications and expressive reasoning across the version space.

The C-OWL formalism presents a definition of so called *context space*, which is build directly on the top of the *OWL space*, together with appropriate extension of the interpretation function [BGvH$^+$03]. Essentially, a context space consists of the set of ontologies in an OWL space and a family of respective mappings. In the scope of multi-version reasoning supported by C-OWL, we will call this structure *mapped version space*, assuming that the version space described in Section 5.1.1 is embedded exactly in the same way as the OWL space in the original definition in [BGvH$^+$03].

The mappings consist of so called *bridge rules*. We adopt these without any modification, since they are directly suitable for setting explicit relations among the particular ontology versions[1]. The bridge rules between elements (concepts, roles or individuals) $i : x, j : y$ from ontologies O_i, O_j, respectively, can be intuitively described as follows (with all the relevant formal definitions given in [BGvH$^+$03] again):

- $i : x \ \rightarrow^{\sqsubseteq} \ j : y$ – states that $i : x$ is more specific than $j : y$
- $i : x \ \rightarrow^{\sqsupseteq} \ j : y$ – states that $i : y$ is more specific than $j : x$
- $i : x \ \rightarrow^{\equiv} \ j : y$ – states that $i : x$ and $j : y$ are equivalent
- $i : x \ \rightarrow^{\perp} \ j : y$ – states that $i : x$ and $j : y$ are incompatible (i.e., their interpretations are disjunct)
- $i : x \ \rightarrow^{*} \ j : y$ – states that $i : x$ and $j : y$ are incompatible (i.e., the intersection of their interpretations is non-empty)

These rules allow us to very naturally model several different relations between elements of ontology versions in a version space. We describe the principles of reasoning with such inter-related versioned ontologies in the next section, provided also by simple illustrative examples.

5.2 Mapped Version Spaces – Inference Issues and an Example

First, we describe essential features of a reasoning engine implementation dealing with bridged ontology (version) spaces in Section 5.2.1. Then we illustrate the proposed modelling of versioned ontologies and its exploitation on an example in Section 5.2.2.

[1]Note that the bridge rules also explicitly identify the foreign concepts for an ontology version, as demanded by the third requirement in the previous section.

5.2.1 Realisation of the Reasoning

Inference in C-OWL is based on so called DDL (Distributed Description Logics) formalism. Distributed DLs are extensions of DLs in which several local ontologies are considered to be related through semantic mappings [BS02a]. DDLs were formally investigated in [BS02b] and are closely related to the multiple viewpoint knowledge representation and reasoning, as described in [GG01].

Modeling and formalizing a multiple viewpoint representation within a DDL is a *decentralized* task, i.e. there is no need to set up a consensus, but, dually, to distinguish viewpoints [dLN07]. Three main steps can be considered:

1. *determine the relevant viewpoints* in the domain,
2. *build a local ontology* for every viewpoint, and
3. *establish mappings* between local ontologies, reifying correspondences between viewpoints.

In the multi-version ontology reasoning settings, we do not have to pay attention to the first two steps in this process – the viewpoints and respective ontologies are implicitly given by the ontology versions. We only have to define the mappings in order to make use of the contextual inference applied to multi-version ontology reasoning.

Local and Global Reasoning Services

Local reasoning services in DDL are the standard DL reasoning services [BCM+03a], performed in a particular context, without taking into account the bridge rules. A *global reasoning service* takes advantage of bridge rules for inferring statements in a context in using knowledge from other contexts. The paper [ST04] presents an extension of the standard tableau algorithm for the computation of the global subsumption test in DDLs. *Global subsumption* relies on the principle of "subsumption propagation" that, in its simplest form, can be expressed as:

if the mapping $\mathcal{M}_{ij}$ contains "$\mathtt{i} : \mathtt{E} \xrightarrow{\sqsupseteq} \mathtt{j} : \mathtt{C}$" and "$\mathtt{i} : \mathtt{F} \xrightarrow{\sqsubseteq} \mathtt{j} : \mathtt{D}$"
then "$\mathfrak{I}$ satisfies $\mathtt{i} : \mathtt{E} \sqsubseteq \mathtt{F}$" implies that "$\mathfrak{I}$ satisfies $\mathtt{j} : \mathtt{C} \sqsubseteq \mathtt{D}$"

Intuitively, this means that subsumption in a particular context can be inferred from subsumption in another context thanks to bridge rules. Similarly, *global instance checking* is based on an instantiation propagation rule:

if $\mathcal{M}_{ij}$ includes "$\mathtt{i} : \mathtt{C} \xrightarrow{\sqsubseteq} \mathtt{j} : \mathtt{D}$" and "$\mathtt{i} : \mathtt{a} \xrightarrow{\equiv} \mathtt{j} : \mathtt{b}$"
then "$\mathfrak{I}$ satisfies $\mathtt{i} : \mathtt{C}(\mathtt{a})$" implies that "$\mathfrak{I}$ satisfies $\mathtt{j} : \mathtt{D}(\mathtt{b})$"

Instantiation is extended for global instance checking. Based on bridge rules, information known about an individual in a particular context can be completed using inferences made in other contexts.

Solving a problem in a multi-viewpoint framework is a *decentralized* process: inferences occur locally, in each viewpoint represented by a context in DDL, taking advantage of bridge rules for reusing the knowledge and inferences from the other viewpoints. This problem-solving approach is similar to a reasoning process in *decentralized artificial intelligence*, defined in [DM89] as being concerned by the activity of autonomous intelligent agents that coexist and collaborate with other agents, each agent having proper goals and proper knowledge.

Implementation of Reasoning in C-OWL

Recently, a KASIMIR system [dBB+05, dLN06, dLN07] has been developed. The system is aimed at decision knowledge management in oncology. The KASIMIR system can be viewed as an intelligent assistant for physicians in their everyday practice of decision making. A novel feature of the research on the KASIMIR system is the use of viewpoints – one viewpoint per an oncology specialty – based on a distributed DL formalism, namely C-OWL. The support for C-OWL in the system is quite general, even though the overall aim of the system is rather specific to e-health applications. As such, the C-OWL reasoning services implemented in KASIMIR can be very naturally used within the multi-version reasoning approach introduced in this chapter.

The architecture of the KASIMIR semantic portal relies on a knowledge server, implemented as a set of Web services and embedding the PELLET OWL reasoner [SPG+07], the JENA API [McB02], and the DRAGO DDL reasoner [ST05]. More details on the KASIMIR system implementation can be found in [dBB+05, dLN06, dLN07].

Querying Version Space Modelled in C-OWL

When modelling the mapped version space using C-OWL as described in this chapter, we can eventually employ multi-version reasoning using the following query templates (evaluated by a C-OWL inference engine):

- check the satisfiability of a given set of statements w.r.t. to a particular ontology version
- check the satisfiability of a given set of statements, iterating through all the versions in a version space (figuring out which versions entail the set in the query)
- determine a version of an ontology element $i : x$ in an ontology version O_j (i.e., find respective equivalent concept(s), if present)

Also, certain validation tasks can be performed:

- determine, whether a given set of statements is compatible (making use of the (in)compatibility bridge rules) w.r.t. a particular version (and its predecessors)
- determine a version (or a version set), which is compatible with a given set of statements (dual task to the previous one)

5.2.2 Example

Imagine a user who builds and maintains an ontology of his or her musical collection (e.g., in order to properly annotate the resources on a desktop). A sample of the initial model (i.e., version O_1) is given in Figure 5.1[2].

Let us assume now that the user adds and retracts some statements, creating another version O_2 from the ontology O_1, as displayed in Figure 5.2.

Two statements related to classical music and one statement related to rock music were added. The class `ex:MTVCrap` was removed, considered perhaps too subjectively motivated now. Moreover, the user may find appropriate to state that punk can also be considered as mainstream in case of some bands. However, he or she does not want to create an explicit sub-class relationship in the current version of the ontology – so he or she uses a compatibility bridge rule, relating punk and mainstream concept defined in the previous version.

Furthermore, the user may want to prevent mixing the semantics of `ex:hasPerformer` and `ex:hasComposer` roles (may possible due to common super-class of the domain classes). However, he or she does not want to impose explicit disjointness restriction here (which would only be possible in OWL 1.1., anyway [HKS05]), since the definition may be still unstable in the scope of future development of the ontology. So he or she uses an incompatibility rule for the respective statements. These rules form the mapping between the versions O_2 and O_1, as displayed in Figure 5.3.

In yet another iteration of the ontology maintenance process, the user changes the class `ex:Punk` into `ex:PunkRock`, however, with no shift in the actual semantics of the class intended. He or she also adds a new individual, `ex:GreenDay`, primarily attributed as a punk playing band. The changes are given in Figure 5.4.

The fact that the semantics of the `ex:Punk` and `ex:PunkRock` classes remains the same can be encoded in a mapping between O_3 and O_2, as shown in Figure 5.5.

Moreover, the user may want to keep a record of the fact that Green Day is not a genuine punk rock band in his or her opinion, and attribute it to the `MTVCrapBand` class in the O_1 version. Although the class does not exist anymore in O_3, there may still be some (possibly shared) annotations of the user's music collection using the old version of

[2]Note that in order to make the presentation as simple as possible, we abstract from exhaustive namespace definitions and additional ontology annotations here, unless found absolutely necessary.

```
<owl:Class rdf:ID="ex:Genre"/>
<owl:Class rdf:ID="ex:Person"/>
<owl:Class rdf:ID="ex:GroupOfPersons"/>
<owl:Class rdf:ID="ex:Rock">
  <rdfs:subClassOf rdf:resource="ex:Genre"/>
</owl:Class>
<owl:Class rdf:ID="ex:Classics">
  <rdfs:subClassOf rdf:resource="ex:Genre"/>
</owl:Class>
<owl:Class rdf:ID="ex:Indie">
  <rdfs:subClassOf rdf:resource="ex:Rock"/>
</owl:Class>
<owl:Class rdf:ID="ex:Mainstream">
  <rdfs:subClassOf rdf:resource="ex:Rock"/>
</owl:Class>
<owl:Class rdf:ID="ex:MTVCrap">
  <rdfs:subClassOf rdf:resource="ex:Mainstream"/>
</owl:Class>
<owl:Class rdf:ID="ex:Neo-Romantic">
  <rdfs:subClassOf rdf:resource="ex:Classics"/>
</owl:Class>
<owl:Class rdf:ID="ex:Minimalist">
  <rdfs:subClassOf rdf:resource="ex:Classics"/>
</owl:Class>
<owl:Class rdf:ID="ex:Band">
  <rdfs:subClassOf rdf:resource="ex:GroupOfPersons"/>
</owl:Class>
<owl:Class rdf:ID="ex:Composer">
  <rdfs:subClassOf rdf:resource="ex:Person"/>
</owl:Class>
<owl:ObjectProperty rdf:ID="ex:hasPerformer">
 <rdfs:domain rdf:resource="ex:Band"/>
 <rdfs:range rdf:resource="ex:Person"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="ex:plays">
 <rdfs:domain rdf:resource="ex:Band"/>
 <rdfs:range rdf:resource="ex:Genre"/>
</owl:ObjectProperty>
<owl:Class rdf:ID="ex:MTVCrapBand"
  <owl:equivalentClass>
   <owl:Restriction>
    <owl:onProperty rdf:resource="ex:plays"/>
    <owl:allValuesFrom rdf:resource="ex:MTVCrap"/>
   </owl:Restriction>
  </owl:equivalentClass>
</owl:Class
```

Figure 5.1: Sample of the initial ontology (O_1)

Added statements:

```
<owl:Class rdf:ID="ex:Punk">
  <rdfs:subClassOf rdf:resource="ex:Indie"/>
</owl:Class>
<owl:Class rdf:ID="ex:MusicalSchool">
  <rdfs:subClassOf rdf:resource="ex:GroupOfPersons"/>
</owl:Class>
<owl:ObjectProperty rdf:ID="ex:hasComposer">
 <rdfs:domain rdf:resource="ex:MusicalSchool"/>
 <rdfs:range rdf:resource="ex:Person"/>
</owl:ObjectProperty>
```

Retracted statements:

```
<owl:Class rdf:ID="ex:MTVCrap">
  <rdfs:subClassOf rdf:resource="ex:Mainstream"/>
</owl:Class>
```

Figure 5.2: Changes transforming O_1 into O_2

```
<cowl:sourceOntology rdf:resource="ex:O2"/>
<cowl:targetOntology rdf:resource="ex:O1"/>

<cowl:bridgeRule cowl:br-type="compat">
 <cowl:sourceConcept rdf:resource="ex:Punk"/>
 <cowl:targetConcept rdf:resource="ex:Mainstream"/>
</cowl:bridgeRule>
<cowl:bridgeRule cowl:br-type="incompat">
 <cowl:sourceConcept rdf:resource="ex:hasComposer"/>
 <cowl:targetConcept rdf:resource="ex:hasPerformer"/>
</cowl:bridgeRule>
```

Figure 5.3: Mapping between O_2 and O_1

Added statements:

```
<owl:Class rdf:ID="ex:PunkRock">
  <rdfs:subClassOf rdf:resource="ex:Indie"/>
</owl:Class>
<ex:PunkRock rdf:ID="ex:GreenDay"/>
```

Retracted statements:

```
<owl:Class rdf:ID="ex:Punk">
  <rdfs:subClassOf rdf:resource="ex:Indie"/>
</owl:Class>
<owl:Class rdf:ID="ex:Punk"/>
```

Figure 5.4: Changes transforming O_2 into O_3

```
<cowl:sourceOntology rdf:resource="ex:O3"/>
<cowl:targetOntology rdf:resource="ex:O2"/>

<cowl:bridgeRule cowl:br-type="equiv">
 <cowl:sourceConcept rdf:resource="ex:PunkRock"/>
 <cowl:targetConcept rdf:resource="ex:Punk"/>
</cowl:bridgeRule>
```

Figure 5.5: Mapping between O_3 and O_2

```
<cowl:sourceOntology rdf:resource="ex:O3"/>
<cowl:targetOntology rdf:resource="ex:O1"/>

<cowl:bridgeRule cowl:br-type="into">
  <cowl:sourceConcept rdf:resource="ex:GreenDay"/>
  <cowl:targetConcept rdf:resource="ex:MTVCrapBand"/>
</cowl:bridgeRule>
```

Figure 5.6: Mapping between O_3 and O_1

the ontology, therefore this mapping may prove useful (e.g. for resource matching). This relation is established using the mapping displayed in Figure 5.6.

Having the musical annotation ontology versions modelled using C-OWL in line with the previous examples, the user can execute for instance the following illustrative queries (using the underlying C-OWL inference engine):

- find a representation of class `ex:Punk` in the version O_3 (result: `ex:PunkRock` in O_3)
- find subclasses of class equivalent to `ex:PunkRock` in the version O_2 (result: subclasses of the `ex:Punk` class in the O_2 version)
- find all bands playing MTV crap music in the versions O_2 and higher (result: `ex:GreenDay` in the sample of the version O_2 in our example)
- validate all definitions of composer individuals in the versions O_2 and higher (i.e., check, whether the definitions using the `ex:hasComposer` role do not interfere with the definitions using the `ex:hasPerformer` role – results in raising an error iff the sets corresponding to the role interpretations are not disjoint)
- ...

Chapter 6

Discussion, Conclusions and Future Work

In this chapter, we give a discussion of the presented topics, provide a conclusion and list the main future tasks that follow from the the content of this report.

6.1 Discussion of the Presented Topics

The content of Chapter 2 and 3 is rather theoretical, however, there are certain consequences for the practice of ontology maintenance, too. We briefly discuss some of the major consequences in rather informal way in Section 6.1.1.

Chapters 4 and 5 present two alternatives of multi-version reasoning. In Section 6.1.2 we show the relation between them, allowing for their possible combination within a practical deployment.

6.1.1 Rational Dynamic Ontology Maintenance

Negation and revision in changing ontologies

Since negation is very closely related to inconsistencies, as shown in Section 2.2, it is very important to consider this fact within the process of ontology maintenance. More specifically, unconsidered introduction of negation can cause incoherence or even inconsistency (see Section 2.2 and 2.3) of the new version of ontology being extended.

In case of OWL (DL) ontologies, care should be exhibited when handling `owl:complementOf` and `owl:disjointWith` statements as a part of the revision. As can be seen in [BvHH+04], these correspond to negation either directly, or via application of de Morgan laws. In an ontology revision implementation, it should be checked whether these particular types of statements in the revision set do not violate consistency, coherency or

any application-specific restrictions when added to the current version of the ontology. Typically, this would be arranged using an associated inference engine. When there is any problem detected, either in the revision set, or in the current ontology, or in both, the involved statements should be adapted in order to prevent the problem (possibly in a way similar to the techniques presented in [HvHH+05]).

In [NLHZ07] we describe an implementation of semi-automatic ontology integration method as a crucial part of the dynamic ontology lifecycle scenario we have introduced in [NHL+06]. This integration can be understood as an ontology revision operator and thus it forms an example of a concrete application that should benefit from following the theoretical notions specified here. In a special section in [NLHZ07] we show, how the "integration operator" conforms to the postulates specified here in Section 2.3.

Contraction in changing ontologies

Implementation of a statement deletion in DL-based ontologies should follow the contraction postulates defined in Section 2.3. This ensures rationality of the behaviour of this type of change (besides trivial removal of all axioms related to statements being removed, of course).

Maintaining ontologies using diffs

Many ontology maintenance implementations use an ontology diff notion and structure quite often [NM04, KFKO02, VG06], however, without a proper and/or common underlying formalisation. The preliminary semantic diff formalisation efforts initiated here in Chapter 3 can help in order to make the diff implementation well-defined and universal even across different application scenarios or (logics-based) ontology languages.

Commitment to a particular practical (weak) diff formal definition ensures well-defined change introduction into evolving ontologies, with clearly understood and efficiently predictable consequences. Quite obviously, this commitment can be combined with conformance to the ontology change postulates from Chapter 2 in order to adopt a well-founded practical ontology change policy in an implementation of ontology maintenance.

6.1.2 Combined Approach to Multi-Version Reasoning

In the previous section, we presented several notes on relation between the introduced theory and practical applications concerning dynamic ontology maintenance and development. In the following, we comment on the issues related to dynamic ontology utilisation – i.e., multi-version reasoning and possibilities of practical combination of the presented approaches.

The C-OWL based reasoning across multiple ontology versions can be easily transformed to the temporal logics-based approach. This is essentially done by omitting the

bridge rules (i.e., the mappings associated to particular ontology versions, relating them to one or more previous versions in the mapped version space; see Section 5.1.2 for details). We end up with a version space (defined in Section 4.1.1) then. The information on the sequence of versions has not changed, therefore we may utilise the model-checking based approach in order to evaluate queries specified in Section 4.2.

Note that since both presented approaches to multi-version reasoning build on the same underlying (description) logics-based inference, the basic queries on concept satisfiability they are able to evaluate are mutually reducible to each other. The only difference is that the C-OWL based approach provides additional query expressivity due to the mapping rules, which have to be explicitly defined.

When it is not feasible to define and maintain the mappings between the versions, the temporal logics-based approach should be the choice for that particular application scenario. When the mappings can be maintained and exploited, combination of both approaches may be useful. Some types of queries can be answered by model-checking and other types (certainly the ones involving bridge rule constructs) using the C-OWL approach. For queries that can be evaluated by both approaches, users may typically choose the more efficient approach.

6.2 Conclusions

We presented a report summarising several theoretical principles that can be used for dealing with changing ontologies. First, formalisation of ontology change was introduced. Chapter 2 dealt with uniform groundwork for negation and change operators in ontologies, adapting the AGM principles [AGM85] in the context of Description Logics. Chapter 3 introduced a preliminary formalisation of logics-based ontology diffs – a feature used in practical ontology development, but rather under-investigated from the theoretical point of view.

Second, we presented two alternatives for reasoning with changing (versioned) ontologies. Chapter 4 and 5 deals with temporal logics-based and C-OWL inspired approach, respectively. Both of these approaches build on the top of the usual Description Logics ontology reasoning, however, providing additional features that allow to query an ontology across a (linearly sorted) sequence of all its versions in time.

We put these topics into a coherent framework and provided a summary in Section 6.1. This summary pointed out the basic theoretical notions directly applicable for practical implementations of dynamic ontology lifecycle, namely for dynamic ontology maintenance and querying of an ontology version repository. The summary establishes a “point of contact” for developers interested in well-founded dynamic ontology maintenance and inference applications. In [NLHZ07] we show a simple ontology integration application that semi-automatically implements ontology revision operator according to the principles discussed here in Section 6.1.1.

6.3 Future Work

The main amount of future work in the scope of the general purpose of this deliverable consists of bridging the gap between theory and practical applications even more. Concerning ontology change formalisation, the analysis of theoretical principles should be combined with appropriate user studies and use cases in order to define and document respective best practices. This would support direct transfer of the theoretical principles into practice and also help to identify, which parts of the theory are perhaps still not that relevant for practical applications.

Regarding the multi-version reasoning, the proposed approach utilising the C-OWL formalism should be elaborated in more detail and eventually implemented. The combination with the temporal logics-based approach should be investigated then, confronting the resulting application to user experience and demands. For instance, support for branching ontology version space can be further analysed and added if proven useful. Support for non-DL ontology reasoning (e.g., RDFS or rule-based) would also be beneficial in many practical applications.

Bibliography

[AGM85] C.E. Alchourrón, P. Gärdenfors, and D. Makinson. On the logic of theory change: partial meet contraction and revision functions. *Journal of Symbolic Logic*, 50(4):510–530, 1985.

[Akm02] V. Akman. *Context in AI: A Fleeting Overview*. McGraw-Hill (Italy), 2002.

[BCM+03a] Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider. *The Decription Logic Handbook: Theory, implementation, and applications*. Cambridge University Press, Cambridge, USA, 2003.

[BCM+03b] Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003.

[BDSZ02] P. Bouquet, A. Dona, L. Serafini, and S. Zanobini. ConTeXtualized local ontology specification via CTXML. In *Working Notes of the AAAI-02 workshop on Meaning Negotiation*. AAAI Press, 2002.

[BGvH+03] P. Bouquet, F. Giunchiglia, F. van Harmelen, L. Serafini, and H. Stuckenschmidt. C-OWL – contextualizing ontologies. In *Proceedings of ISWC03*. Springer-Verlag, 2003.

[BMC03] S. Bechhofer, R. Moeller, and P. Crowther. The DIG description logic interface. In *Proc. of International Workshop on Description Logics (DL2003)*, 2003.

[BS02a] A. Borgida and L. Serafini. Distributed description logics: Directed domain correspondences in federated information sources. In *Proc. of the Intenational Conference on Cooperative Information Systems*, 2002.

[BS02b] A. Borgida and L. Serafini. Distributed Description Logics: Directed domain correspondences in federated information sources. In R. Meersman and Z. Tari, editors, *On The Move to Meaningful Internet Systems 2002: CoopIS, DOA, and ODBASE Proceedings*, LNCS 2519, pages 36–53. Springer, Berlin, 2002.

[BvHH+04] S. Bechhofer, F. van Harmelen, J. Hendler, I. Horrocks, D. L. McGuinness, P. F. Patel-Schneider, and L. A. Stein. *OWL Web Ontology Language Reference*, 2004. Available at (February 2006): `http://www.w3.org/TR/owl-ref/`.

[CGP99] Edmund M. Clarke, Orna Grumberg, and Doron A. Peled. *Model Checking*. The MIT Press, Cambridge, Massachusetts, 1999.

[dBB+05] M. d'Aquin, S. Brachais, C. Bouthier, J. Lieber, and A. Napoli. Knowledge Editing and Maintenance Tools for a Semantic Portal in Oncology. *International Journal of Human-Computer Studies (IJHCS)*, 62(5):619–638, 2005.

[dLN06] M. d'Aquin, J. Lieber, and A. Napoli. Case-based reasoning within semantic web technologies. In J. Euzenat and J. Domingue, editors, *Proceedings of the 12th International Conference on Artificial Intelligence: Methodology, Systems, Applications (AIMSA 2006), Varna, Bulgaria, 13-15th September, 2006*, LNAI 4183, pages 190–200. Springer, Berlin, 2006.

[dLN07] M. d'Aquin, J. Lieber, and A. Napoli. Applying Advanced Description Logic Reasoning for Decision Support in Oncology. Techincal report, LORIA (Nancy), 2007.

[DM89] Y. Demazeau and J.-P. Mller. Decentralized Artificial Intelligence. In Y. Demazeau and J.-P. Mller, editors, *Decentralized A.I. – Proc. of the First European Workshop on Modelling Autonomous Agents in a Multi-Agent World*, pages 3–13. North-Holland, 1989.

[FHP+06] Giorgos Flouris, Zhisheng Huang, Jeff Z. Pan, Dimitris Plexousakis, and Holger Wache. Inconsistencies, negations and changes in ontologies. In *Proceedings of AAAI 2006*. AAAI Press, 2006.

[Flo06] Giorgos Flouris. *On Belief Change and Ontology Evolution*. PhD thesis, Department of Computer Science, University of Crete, 2006.

[FPA04] Giorgos Flouris, Dimitris Plexousakis, and Grigoris Antoniou. Generalizing the agm postulates: preliminary results and applications. In *Proceedings of the* 10^{th} *International Workshop on Non-Monotonic Reasoning*, pages 171–179, 2004.

[FPA06] Giorgos Flouris, Dimitris Plexousakis, and Grigoris Antoniou. On generalizing the agm postulates. In *Proceedings of the 3rd European Starting AI Researcher Symposium (STAIRS-06)*, pages 132–143, 2006.

[Fuh91] A. Fuhrmann. Theory contraction through base contraction. *Journal of Philosophical Logic*, 20:175–203, 1991.

[Gär92] P. Gärdenfors. Belief revision. In P. Gärdenfors, editor, *Belief Revision: An Introduction*, pages 1–20. Cambridge University Press, 1992.

[GG01] C. Ghidini and F. Giunchiglia. Local models semantics, or contextual reasoning = locality + compatibility. *Artificial Intelligence*, 127, 2001.

[HKS05] I. Horrocks, O. Kutz, and U. Sattler. The even more irresistible $\mathcal{SROIQ}$. Technical report, University of Manchester, 2005.

[HP04] J. Heflin and Z. Pan. A model theoretic semantics for ontology versioning. In *Proceedings of ISWC2004*, pages 62–76, Hiroshima, Japan, 2004. Springer.

[HPS03] Ian Horrocks and Peter F. Patel-Schneider. Reducing OWL entailment to description logic satisfiability. In *Proc. of the 2003 International Semantic Web Conference (ISWC 2003)*, pages 17–29, 2003.

[HS05a] Zhisheng Huang and Heiner Stuckenschmidt. Reasoning with multi-version ontologies. Technical Report 351, SEKT, 2005.

[HS05b] Zhisheng Huang and Heiner Stuckenschmidt. Reasoning with multi-version ontologies: A temporal logics approach. In *Proceedings of ISWC'05*. Springer-Verlag, 2005.

[HST00] I. Horrocks, U. Sattler, and S. Tobies. Practical Reasoning for Very Expressive Description Logics. *Logic Journal of the IGPL*, 8(3):239–263, 2000.

[HSvHK06] Zhisheng Huang, Stefan Schlobach, Frank van Harmelen, and Michel Klein. Reasoning with multi-version ontologies: Evaluation. Technical Report 352, SEKT, 2006.

[HvHH+05] Peter Haase, Frank van Harmelen, Zhisheng Huang, Heiner Stuckenschmidt, and York Sure. A framework for handling inconsistency in changing ontologies. In *Proceedings of the 2005 International Semantic Web Conference (ISWC05)*, 2005.

[HvHtT05] Z. Huang, F. van Harmelen, and A. ten Teije. Reasoning with inconsistent ontologies. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI 2005)*, 2005.

[KF01] Michel Klein and Dieter Fensel. Ontology versioning for the semantic web. In *Proceedings of 1st Int Semantic Web Working Symposium*, pages 75–91, Stanford, CA, USA, 2001. Stanford University.

[KFKO02] Michel Klein, Dieter Fensel, Atanas Kiryakov, and Damyan Ognyanov. Ontology versioning and change detection on the web. In *Proceedings of EKAW 2002*, pages 197–212, Berlin, 2002. Springer–Verlag.

[KM91] H. Katsuno and A. Mendelzon. On the difference between updating a knowledge base and revising it. In *Proceedings of KR'91*, pages 387–394, 1991.

[McB02] B. McBride. Jena: a semantic Web toolkit. *Internet Computing, IEEE*, 6(6):55–59, 2002.

[McC93] J. McCarthy. Formalizing context. In *Proceedings of IJCAI'93*. Morgan Kauffman, 1993.

[NHL+06] Vít Nováček, Siegfried Handschuh, Loredana Laera, Diana Maynard, Max Völkel, Tudor Groza, Valentina Tamma, and Sebastian Ryszard Kruk. Report and prototype of dynamics in the ontology lifecycle (D2.3.8v1). Deliverable 238v1, Knowledge Web, 2006.

[NK04] Natalya F. Noy and Michel Klein. Ontology evolution: Not the same as schema evolution. *Knowledge and Information Systems*, pages 428–440, 2004.

[NLHZ07] Vít Nováček, Loredana Laera, Siegfried Handschuh, and Jan Zemánek. Report and prototype of dynamics in the ontology lifecycle, version 2 (D2.3.8v2). Deliverable 238v2, Knowledge Web, 2007. Available since December 2007.

[NM04] N. F. Noy and M. A. Musen. Ontology versioning in an ontology management framework. *IEEE Intelligent Systems*, 19(4):6–13, 2004.

[PSHH04] Peter F. Patel-Schneider, Patrick Hayes, and Ian Horrocks. OWL Web Ontology Language Semantics and Abstract Syntax. Technical report, W3C, Feb. 2004. W3C Recommendation.

[RU71] N. Rescher and A. Urquhart. *Temporal Logic*. Springer-Verlag, 1971.

[SC03] S. Schlobach and R. Cornet. Non-standard reasoning services for the debugging of description logic terminologies. In *Proceedings of IJCAI2003*, Acapulco, Mexico, 2003. Morgan Kaufmann.

[SPG+07] E. Sirin, B. Parsia, B. Cuenca Grau, A. Kalyanpur, and Y. Katz. Pellet: A practical owl-dl reasoner. *Journal of Web Semantics*, 2007. (To Appear).

[ST04] L. Serafini and A. Tamilin. Local Tableaux for Reasoning in Distributed Description Logics. In V. Haarslev and R. Moeller, editors, *Proceedings of the International Workshop on Description Logics, DL'04, Whistler, BC, Canada*, pages 100–109. CEUR-WS, Volume 104, 2004.

[ST05] L. Serafini and A. Tamilin. DRAGO: Distributed reasoning architecture for the semantic web. In A. Gomez-Perez and J. Euzenat, editors, *Proc. of the Second European Semantic Web Conference (ESWC'05)*, volume 3532 of *Lecture Notes in Computer Science*, pages 361–376. Springer, 2005.

[Sto04] L. Stojanovic. *Methods and Tools for Ontology Evolution*. PhD thesis, University of Karlsruhe, 2004.

[UG96] M. Uschold and M. Gruninger. Ontologies: Principles, Methods and Applications. *The Knowledge Engineering Review*, 1996.

[vB95] Johan van Benthem. Temporal logic. In *Handbook of Logic in Artificial Intelligence and Logic Programming*, volume 4, pages 241–350. Oxford, Clarendon Press, 1995.

[VEK+05] M. Völkel, C. F. Enguix, S. R. Kruk, A. V. Zhdanova, R. Stevens, and Y. Sure. SemVersion – versioning RDF and ontologies (D2.3.3v1). Deliverable 233v1, Knowledge Web, 2005.

[VG06] Max Völkel and Tudor Groza. SemVersion: RDF-based ontology versioning system. In *Proceedings of the IADIS International Conference WWW/Internet 2006 (ICWI 2006)*, 2006.

[VKZ+05] M. Völkel, S. R. Kruk, A. V. Zhdanova, R. Stevens, A. Artale, E. Franconi, and S. Tessaris. SemVersion – versioning RDF and ontologies (D2.3.3v2). Deliverable 233v2, Knowledge Web, 2005.

Related Deliverables

The work presented here is directly related to the following deliverables:

Project	Number	**Title** and relationship
KW	D2.3.3v1	**SemVersion – Versioning RDF and Ontologies (D2.3.3v1)** – ontology versioning methodology proposal and implementation
KW	D2.3.3v2	**SemVersion – Versioning RDF and Ontologies (D2.3.3v2)** – ontology versioning methodology proposal, implementation and evaluation
KW	D2.3.8v1	**Report and Prototype of Dynamics in the Ontology Lifecycle (D2.3.8v1)** – proposal of dynamic ontology lifecycle scenario
KW	D2.3.8v2	**Report and Prototype of Dynamics in the Ontology Lifecycle (D2.3.8v2)** – proposal, implementation and basic evaluation of a dynamic ontology learning and integration prototype, designed in line with the scenario defined in D2.3.8v1 and the principles introduced here
SEKT	D3.5.1	**Reasoning with Multi-version Ontologies** – a report dealing with details of temporal logics-based multi-version reasoning